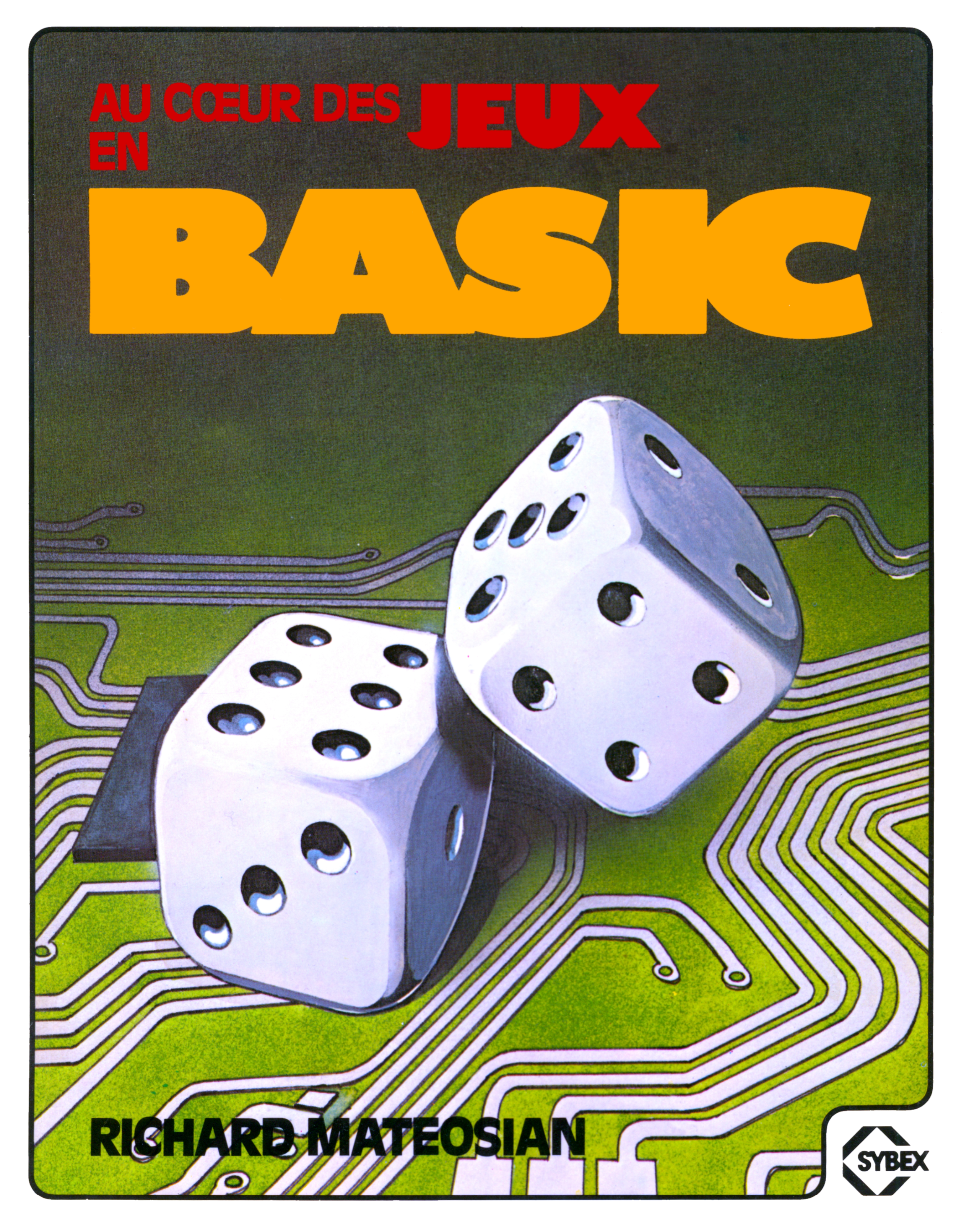

AU CŒUR DES JEUX
EN
BASIC
RICHARD MATEOSIAN
SYBEX

AU CŒUR DES JEUX EN BASIC

AU CŒUR DES JEUX EN BASIC

Richard Mateosian



Traduction Française : Bernard Besse.

NOTE

Apple est une marque déposée de Apple Computer Inc.
TRS-80 est une marque déposée de Tandy Corporation.
PET/CBM sont des marques déposées de Commodore Inc.
Dans le présent ouvrage tous ces noms doivent être pris adjectivement.

Couverture de Daniel Le Noury

Illustrations techniques de J. Trujillo Smith, Jeanne E. Tennant.

Tous les efforts ont été faits pour fournir dans ce livre, une information complète, et exacte. Néanmoins, SYBEX n'assume de responsabilités ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

ISBN : 2-902414-33-1

Table des matières

Préface

Les trois principaux thèmes de ce livre sont :

— La formation par les jeux.
— La programmation méthodique en utilisant le BASIC Libre.
— Les jeux en BASIC sur ordinateur.

Le premier thème est une conséquence directe de la suggestion que Rodnay Zaks me fit d'écrire un livre sur les jeux en BASIC. « Non, pas simplement un recueil de jeux, mais un ouvrage d'enseignement qui aiderait le lecteur à concevoir ses propres programmes en BASIC », tel fut l'objectif indiqué. Tout lecteur qui étudie le contenu de *Au cœur des Jeux en BASIC*, sera effectivement capable de réaliser des programmes BASIC soit pour des jeux, soit pour d'autres applications.

Le second thème vient de l'expérience que j'ai acquise en 1975, en réalisant plusieurs applications de moyenne importance sur un DTC Microfile, l'un des premiers micro-ordinateurs assez semblable sur le plan fonctionnel au TRS-80 à disques. A cette époque, je développais la plupart des idées figurant dans ce livre, et notamment, un précurseur du BASIC Libre.

L'origine des jeux est variée. J'ai écrit tous les programmes moi-même, à partir de zéro. Les jeux que j'ai inventés pour ce livre sont : l'Entraînement à l'Arithmétique, l'Horloge, la Mémoire des Cartes, le Chronomètre, l'Anniversaire, le Jumelage et la Rencontre avec les Extra-Terrestres. Les autres sont des jeux sur ordinateur très connus, bien qu'on ignore souvent leur origine.

Donner vie à un livre est une tâche longue et difficile, mais enrichissante. Un des plaisirs réels donnés à l'auteur est l'occasion qui lui est offerte de remercier publiquement quelques-uns de ceux qui ont contribué à l'œuvre finale.

Et tout d'abord, j'exprime ma reconnaissance à Rudolph Langer, Editeur en chef de SYBEX, pour le haut niveau de qualité dont l'ouvrage a bénéficié.

De façon plus précise, je désire remercier le département éditorial de SYBEX, pour avoir poli l'ouvrage, résolu les ambiguïtés, fait la chasse aux contradictions, extirpé, avec mon accord, ce qui leur paraissait hors du sujet, et pour avoir transformé un manuscrit mal dégrossi en un texte définitif. Quelques-unes de leurs améliorations ont été véritablement des coups de génie.

Ce livre a tiré grand profit des capacités artistiques de J. Trujillo Smith qui s'est chargé des illustrations et de leur mise en page.

Le remerciement final doit aller à ma femme, Virginia Ruth Mason. Je n'aurais pas écrit cet ouvrage sans sa patience, sa compréhension et ses sacrifices.

Berkeley, California
Février 1981

Introduction

Ce livre a été écrit pour tous ceux qui veulent comprendre le fonctionnement des jeux sur ordinateur et apprendre à écrire en BASIC des programmes conversationnels. Nous verrons ensemble :

— Comment on construit un programme conversationnel, et de quelle façon les principes de développement des systèmes sont appliqués sur micro-ordinateurs.

— Comment les caractéristiques de certains micro-ordinateurs ont été prises en compte dans le langage BASIC.

— Comment le BASIC Libre peut vous aider à concevoir les programmes BASIC de façon méthodique.

Les jeux ont été choisis comme exemples de programmation pour plusieurs raisons :

— Les jeux intéressent en priorité de nombreux utilisateurs et programmeurs de micro-ordinateurs.

— Les jeux ne demandent pas de connaissances particulières pour être compris.

— Les jeux mettent en lumière la plupart des cas particuliers de programmation que l'on trouve dans les autres applications conversationnelles.

Les lecteurs de ce livre se rangeront en grande partie dans l'une des catégories suivantes :

— Ceux qui n'ont programmé que sur micro-ordinateurs.

— Ceux qui ont programmé sur « gros systèmes », ou, sur minis, et connaissent au moins un langage évolué, (et peut-être un langage assembleur).

Si vous n'avez jamais écrit de programme avant de devenir propriétaire de votre ordinateur individuel, vous avez probablement lu le manuel livré avec la machine et passé de nombreuses heures à étudier les exemples présentés. Ensuite, vous vous êtes vraisemblablement procuré des programmes auprès de vendeurs de logiciel, et des livres traitant de jeux BASIC ou de programmes plus « sérieux ». Peut-être même avez-vous lu un ouvrage sur la programmation BASIC ou sur l'art de programmer.

Ce livre vous permettra de mieux comprendre l'informatique, le BASIC et les particularités de votre ordinateur individuel. Il vous apportera :

— Des commentaires détaillés pour vous guider tout au long des programmes BASIC réels.
— L'explication des techniques de structuration des programmes, de codage de l'information ainsi que la description des algorithmes couramment utilisés en programmation.
— Une description détaillée du BASIC Libre – technique de conception des programmes qui vous aidera à pallier une grande partie des inconvénients du BASIC.

Si vous êtes un programmeur expérimenté, devenu propriétaire d'un ordinateur individuel, vous connaissez déjà les principes de fonctionnement des ordinateurs et vous savez à quoi servent les langages de programmation. Ce livre vous montrera les points faibles de votre nouveau domaine de programmation, et vous présentera :

— Les caractéristiques du BASIC et celles de votre ordinateur.
— Un puissant outil de conception des programmes, le BASIC Libre.
— Des exemples des méthodes spécifiques de programmation interactives des petits systèmes.

Le même plan sera suivi pour tout ce qui concerne les jeux. Premièrement, nous exposerons les règles du jeu ; deuxièmement, nous présenterons et commenterons le programme utilisé pour la réalisation de ce jeu. Ces commentaires seront centrés sur les principes et les techniques de la programmation des jeux en BASIC.

Le « Menu »

— ***Les Jeux Arithmétiques.*** Nous verrons d'abord un jeu pédagogique très simple : « L'Entraînement à l'Addition ». La généralisation, l'un des concepts les plus importants en programmation, est alors utilisée pour développer un programme plus complexe : « L'Entraînement à l'Arithmétique ».

— ***Les Jeux de la Découverte.*** Ce sont des jeux portant sur la recherche d'un nombre, et appelés, Un, Deux, Trois, Quatre, ..., Neuf, ainsi que deux jeux portant sur la recherche d'un mot. *Le Mot et le Pendu*. On applique de nouveau le principe de

généralisation pour faire tenir ces deux jeux en un seul programme. Une technique de « cannibalisation » facile à mettre en œuvre, permet d'obtenir *Le Pendu* à partir de ce dernier. L'analyse de ces jeux donne également un exemple de la stratégie appliquée par un bon joueur de *Quatre*.

— ***Les Jeux Horaires.*** Ce chapitre expose les différentes techniques utilisées pour traiter l'élément temps dans les jeux et autres applications. Les quatre programmes présentés sont *L'Horloge, La Mémoire des Cartes, le Flicker à Dix Touches* et *le Chronomètre*. La structuration des programmes et l'utilisation de « souches » sont décrites pour la mise en pratique de la démarche descendante. Parmi les techniques de programmation étudiées figurent la synchronisation des événements avec les « battements » de l'horloge, l'utilisation du temps « Julien », ainsi que la génération de délais d'une durée donnée.

— ***Les Jeux Chronologiques.*** Ils réunissent les jeux *Anniversaire* et *Calendrier*. *Anniversaire* montre comment on trouve le jour de la semaine d'une date quelconque. *Calendrier* dessine sur l'écran le calendrier correspondant à un mois et une année donnés. L'importance du positionnement du curseur pour réaliser cette image y est exposée.

— ***Le Fisc*** est un jeu mystérieux qui vous met au défi de trouver les règles qu'il applique. (Et même lorsque vous les aurez découvertes, le jeu continuera à vous défier.) Le programme correspondant tient compte des particularités des différents systèmes BASIC pour traiter les chaînes de chiffres.

— ***La Programmation en BASIC Libre.*** Il s'agit d'une technique qui facilite la conception des programmes BASIC. Le BASIC Libre est un BASIC « structuré », traduit ensuite par le programmeur en instructions BASIC proprement dites, destinées à être entrées en machine. Le BASIC Libre n'est pas un langage ; c'est un moyen de décrire le programme (comme le sont les organigrammes). Une description de programme en BASIC Libre n'a aucun numéro de ligne et utilise des noms symboliques pour désigner les sous-programmes. Les instructions GOTO en sont exclues (vous n'en trouverez aucune dans les descriptions en BASIC Libre du présent ouvrage), et les structures de contrôle du type *if ... then ... else, repeat ... until* et *while* sont utilisées à leur place.

Tous les programmes des cinq premiers chapitres sont expliqués par des instructions BASIC, et il n'est pas nécessaire de connaître le BASIC Libre pour les comprendre. Néanmoins, le programmeur expérimenté pourrait avantageusement lire le chapitre *La Programmation en BASIC Libre* pour pouvoir utiliser les descriptions en BASIC Libre qui apparaissent dans ces cinq premiers chapitres.

L'analyse des programmes des trois derniers chapitres repose uniquement sur leurs descriptions en BASIC Libre. Néanmoins, la liste des instructions BASIC correspondante, continuera à être donnée. Vous pourrez vous en servir pour entrer ces programmes à la main dans votre ordinateur. Aux chapitres 7 et 9, les programmes BASIC sont présentés sous la même forme compacte (sans espaces), forme que l'auteur a été contraint d'utiliser pour permettre l'exécution sur son Pet 8K.

— ***Le Jeu du Jumelage*** est commenté sur la base de sa description en BASIC Libre, sans utiliser le BASIC proprement dit. Dans *Jumelage* on doit associer par paires les éléments de deux groupes. Les joueurs ont la possibilité d'adapter le jeu à leurs propres besoins et à leur intérêt personnel. Le programme correspondant à ce jeu est assez volumineux. Il contient plus de quarante sous-programmes et dépasse les 500 lignes de BASIC Libre. Si vous maîtrisez toutes les finesses de ce chapitre, *c'est que vous êtes en mesure de concevoir et de réaliser des systèmes conversationnels en BASIC.*

— ***Craps.*** Ce jeu de dés est plus simple que *Jumelage*, mais diffère sur les points suivants : technique d'affichage sur l'écran, tirage au hasard, choix du traitement en fonction de la situation instantanée, certitude d'un déroulement rapide et harmonieux. Si vous apprenez et mettez en pratique les principes décrits dans ce chapitre, les jeux que vous concevrez seront très amusants.

— ***« Les Extra-Terrestres »*** présente le jeu *Contact avec les Extra-Terrestres*, jeu qui inclut un autre très connu, celui du *Jeu de la Vie*. Les programmes de ce chapitre vous donneront un aperçu des techniques utilisées pour l'affichage en mode graphique sur votre ordinateur individuel.

A quels ordinateurs le contenu de ce livre s'applique-t-il ?

Les programmes ont été conçus pour fonctionner sur n'importe quel ordinateur. Les éléments de programme qu'il faut modifier en

cas de changement de machine, sont peu importants et ont été soigneusement signalés. Chaque programme a été décrit pour fonctionner sur une des configurations de l'un, au moins, des ordinateurs « individuels » suivants :

— Le Système TRS-80 de Radio Shack (BASIC Niveau II).

— Le Pet de Commodore Business Machines.

— Le Modèle II d'Apple Computer (BASIC Applesoft).

CHAPITRE **1**

Jeux arithmétiques

Ce chapitre décrit deux jeux arithmétiques simples, l'Entraînement à l'Addition et l'Entraînement à l'Arithmétique. Le premier jeu est facile, il vous permettra de vous familiariser avec les conventions utilisées dans ce livre. Le deuxième ne diffère que légèrement du premier, il pose un problème plus intéressant. Le processus de création du second programme à partir du premier illustre les principes de la généralisation.

L'Entraînement à l'Addition

Ce jeu se joue de la façon suivante : Le programme commence avec une question du genre :

QUE FONT 1 + 8 ?

Vous répondez en frappant sur le clavier votre solution et vous appuyez sur RETURN (ou ENTER). La somme de 1 et 8 étant 9, vous tapez 9 et appuyez sur RETURN (ENTER). Le programme répondra :

EXACT — ESSAYEZ DONC ENCORE UNE FOIS
QUE FONT 3 + 4 ?

Chaque fois que votre réponse est bonne, le programme vous pose une nouvelle question. Mais supposons que votre doigt dérape et qu'au lieu d'entrer un 7 (suivi de RETURN) vous répondiez par un 8. La réaction du programme sera :

DESOLE, C'EST FAUX — RECOMMENCEZ
QUE FONT 3 + 4 ?

Voilà tout le jeu. Toutes les fois que votre réponse est bonne, le programme vous pose une autre question; lorsque votre réponse est fausse, le programme repose la même question jusqu'à ce que votre réponse soit correcte.

Le programme d'Entraînement à l'Addition

La figure 1 . 1 montre l'organigramme de ce jeu. Pour lire cet organigramme, commencez par le cercle marqué « DEBUT » et suivez

les flèches. Lorsque vous arriverez à la case en forme de losange d'où partent deux flèches, vous suivrez, soit la flèche marquée NON — si la réponse à la question dans ce losange est « non », soit la flèche marquée OUI — si la réponse est « oui ». Ces deux chemins vous ramèneront dans une partie du schéma dans laquelle vous êtes déjà passé. On appelle *boucle*, toute partie d'organigramme qu'on est obligé de parcourir plusieurs fois de suite.

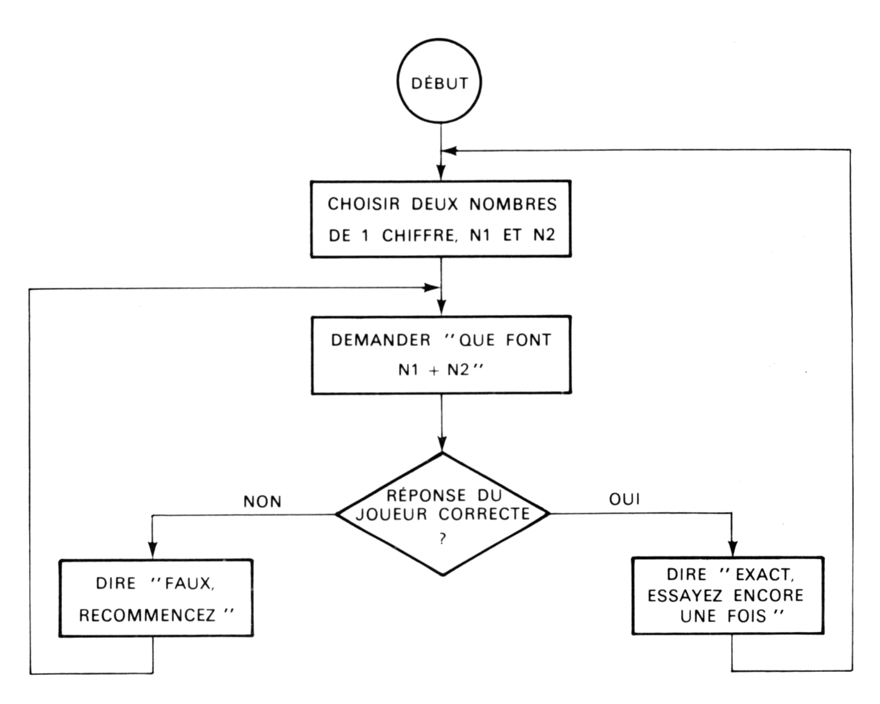


Figure 1.1 : Organigramme de l'Entraînement à l'Addition

La figure 1.2 présente quelques instructions écrites en vrai BASIC qui donneront à l'ordinateur, le comportement que nous venons de décrire. Ces instructions sont très proches de celles qui figurent dans les cases de l'organigramme. Les lignes 100 et 110 correspondent à la première case. Les lignes 120 et 130 correspondent à la seconde case, tandis que la ligne 140 précise un détail d'édition qui n'est pas indiqué dans l'organigramme. La ligne 150 correspond au losange et au chemin marqué « non », tandis que les lignes 160 170 correspondent au chemin marqué « oui ».

Examinons pas à pas le programme de la figure 1.2. L'action commence à la ligne 100 (parce que le BASIC commence toujours par la ligne portant le numéro le plus faible) et se poursuit séquentiellement sauf lorsqu'une instruction spécifique (telle que GOTO 120) vient modifier ce déroulement.

```
100*  N1=INT(RND(1)*10)
110*  N2=INT(RND(1)*10)
120   PRINT"QUE FONT";N1;"+";N2;
130   INPUT A
140   PRINT
150   IF A< >N1 +N2 THEN PRINT "DESOLE, C'EST FAUX — RECOMMENCEZ": GOTO 120
160   PRINT "EXACT—ESSAYEZ DONC ENCORE UNE FOIS"
170   GOTO 100
```

Ce programme tire au hasard deux nombres d'un seul chiffre et demande au joueur quelle est leur somme. Si la réponse est correcte, le programme donne un autre exercice à faire. Si la réponse est fausse, le programme répète la question jusqu'à ce que la réponse soit juste.

Le programme ci-dessus fonctionne sur PET et TRS-80. Sur Apple, la ligne 120 doit être remplacée par :

```
120   PRINT "QUE FONT";N1;"+";N2;"";
```

La différence tient au fait que l'Apple n'insère d'espaces ni avant, ni après les nombres, alors que le PET et le TRS-80 mettent un espace en fin de chaque nombre et aussi en tête des nombres positifs.

* Sur TRS-80, l'expression RND(0) remplace RND(1). Sur ce système tout argument autre que zéro a une signification particulière. Sur Pet et Apple, au contraire, seul RND(0) est spécial.

Figure 1.2 : L'Entraînement à l'Addition

La génération de Nombres au Hasard

Dans ce programme, la ligne 100 contient l'instruction :

N1 = INT(RND(1) * 10)

En gros, cette instruction de la ligne 100 signifie : « Choisissez un nombre entier compris entre zéro et neuf, puis stockez-le dans un « tiroir » portant l'étiquette N1. » Ce nombre est le premier des deux nombres dont le programme va vous demander la somme.

La ligne 110 contient l'instruction :

N2 = INT(RND(1) * 10)

Cette instruction est identique à la première, mais sur la gauche du signe égal, N1 est devenu N2. La ligne 110 veut dire : « Choisissez un autre nombre entier compris entre zéro et neuf et stockez-le dans le tiroir appelé N2. »

Bien que l'expression se trouvant sur la droite du signe égal soit la même dans les deux instructions, les nombres stockés dans les tiroirs N1 et N2 ne sont pas forcément les mêmes. Ce changement de valeur tient à la présence du terme RND(1) dans l'instruction. RND, abréviation de « Random » (au hasard), est le nom de la fonction de génération des nombres au hasard. Lors de l'exécution, l'expression RND(1) génère un nombre « imprévisible » compris entre zéro et un (mais jamais égal à zéro ou à un).

Pour ce jeu, nous avons besoin de nombres entiers égaux ou supérieurs à zéro et inférieurs à dix. On peut faire passer l'intervalle original de zéro-à-un à un intervalle variant de zéro-à-N. en multipliant tout simplement par N le nombre fourni par RND. C'est ainsi que la présence dans chacune des deux instructions de l'expression

```
RND(1) * 10
```

aura pour effet, à chaque exécution, de produire un nombre au hasard compris entre zéro et dix (mais jamais égal à zéro ou à dix).

La fonction INT ne fait que se « débarrasser » de la partie décimale du nombre. C'est ainsi que INT(5.63279) vaut 5, que INT(0.22116) vaut 0 et que INT(9.99999) vaut 9. INT est l'abréviation de *integer*, nom commun anglais signifiant nombre entier. La fonction INT donne comme résultat la partie entière du nombre qui lui est présenté.

Ainsi, l'expression

```
INT(RND(1) * 10)
```

donne (toujours) comme résultat, un nombre entier au hasard pris dans la série : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. (En comprenez-vous la raison ? Pourquoi ce nombre entier n'est-il jamais dix ? Pourquoi peut-il être zéro ?)

L'Instruction PRINT

A la ligne 120, nous avons la troisième instruction :

```
PRINT "QUE FONT "; N1; "+ "; N2;
```

Cette instruction fait afficher par l'ordinateur la question " QUE FONT ____ + ____ ? " (à l'exception du point d'interrogation qui est rajouté par le système lors de l'instruction INPUT). Malheureusement,

ce qui apparaît en fait sur l'écran, dépend de l'ordinateur individuel utilisé. Par exemple sur un Pet, cette instruction donnera le résultat suivant :

QUE FONT 9 + 7

Sur Apple, cela deviendra :

QUE FONT9+7

La différence tient au fait qu'avec le Pet, tout nombre est suivi d'un espace, et, s'il est positif, précédé d'un autre espace. L'Apple n'insère d'espaces ni avant ni après les nombres. (Sur ce point le TRS-80 se comporte comme le Pet.)

Il existe d'autres différences entre les instructions PRINT des ordinateurs individuels. Nous les verrons au fur et à mesure. Mais tout d'abord, occupons-nous du fonctionnement de la forme la plus simple de l'instruction PRINT. (Cette forme simple donne à peu près le même résultat sur tous les systèmes BASIC.) Elle est constituée par le mot PRINT suivi d'une liste de *variables* et de *constantes*.

Par définition, une constante est un nombre, comme 5 ou 9.763, ou un fragment de texte (comme "QUE FONT") placé entre guillemets. Quant à la variable, c'est l'étiquette du tiroir, autrement dit, c'est le nom de l'endroit de la mémoire de l'ordinateur où nous avions auparavant stocké une constante. En termes plus précis, une variable est également appelée *adresse symbolique* puisque c'est le nom donné à une certaine place en mémoire. Dans l'instruction

PRINT "QUE FONT"; N1 ; " + " ; N2;

la liste qui suit le mot PRINT est composée de la constante " QUE FONT ", de la variable N1, de la constante " + " et de la variable N2. Les points-virgules servent à séparer les éléments de la liste et à préciser la façon dont l'affichage devra être effectué sur l'écran.

Le BASIC exécute une instruction PRINT en parcourant la liste des constantes et des variables et en affichant leurs *valeurs*. Donc, chaque fois que dans la liste, le BASIC rencontre une constante, il affiche le fragment de texte ou le nombre en question ; quand il s'agit d'une variable, il affiche le nombre ou le fragment de texte stocké dans le tiroir correspondant. Ainsi, si nos deux premières instructions ont eu pour effet de stocker 5 en N1 et 7 en N2, le résultat de l'instruction

PRINT "QUE FONT"; N1 ; " + " N2;

sera le suivant :

QUE FONT 5 + 7

Nous venons d'expliquer comment, lorsqu'il exécute une instruction PRINT, le BASIC détermine les valeurs à afficher. Nous allons maintenant voir comment le BASIC choisit les positions de l'écran où doivent être affichées les valeurs en question. Dans notre exemple, les points-virgules indiquent, en BASIC, qu'il faut mettre les différentes données « bout à bcut » sans insérer d'espaces supplémentaires ni revenir à la ligne. L'autre signe utilisé pour séparer les éléments d'une liste est la virgule. Lorsqu'un élément de la liste est suivi d'une virgule, le BASIC, après avoir affiché la valeur correspondante, déplace le curseur jusqu'à la *position de tabulation* suivante.

Les positions de tabulation ressemblent à celles que l'on peut régler sur une machine à écrire. Cependant avec le BASIC, elles sont pré-réglées et ne peuvent être modifiées par le programmeur. Votre ordinateur individuel a trois ou quatre positions de tabulation par ligne de 40 caractères. Ainsi, si nous avions écrit

```
PRINT "QUE FONT", N1, "+", N2
```

la sortie affichée aurait été ceci :

```
QUE FONT    5     +     7
```

ou même cela :

```
QUE FONT    5     +
7
```

Dans les exemples ci-dessus, nous n'avons pas vu quel est l'effet du point-virgule ou de la virgule placés en fin de ligne. Pour le savoir, regardons la suite de notre programme. A la ligne 130, l'instruction

```
INPUT A
```

sert à rentrer une information donnée par le joueur. L'instruction INPUT a toujours pour effet l'affichage d'un « ? ». Ainsi, les deux lignes

```
PRINT "QUE FONT"; N1; "+"; N2;
INPUT A
```

entraînent l'affichage suivant :

```
QUE FONT 5 + 7 ?
```

Si nous avions omis le point-virgule final, l'instruction PRINT aurait affiché ceci

```
QUE FONT 5 + 7
?
```

Ces exemples montrent que chaque fois qu'une instruction PRINT ne se termine pas par une virgule ou par un point-virgule, le BASIC conclut l'exécution de l'instruction, en plaçant le curseur au début de la ligne suivante.

Nous avons maintenant passé en revue tous les aspects de la forme la plus simple de l'instruction PRINT. Celle-ci jouant un rôle très important dans la plupart des programmes BASIC, il est nécessaire que vous la connaissiez parfaitement. Et maintenant, essayez de prévoir ce qui sera affiché par les instructions

```
PRINT "QUE FONT", N1, " + ", N2
INPUT A
```

vérifiez ensuite votre prévision sur votre ordinateur individuel.

L'Instruction INPUT

Revenons à notre programme. L'instruction BASIC

```
INPUT A
```

affiche " ? " (ou " ? " sans espace sur Apple) et ensuite attend que vous ayez tapé un nombre et appuyé sur la touche RETURN (ou ENTER sur TRS-80). Le BASIC s'assure ensuite que c'est bien un nombre qui a été entré ; si ce n'est pas le cas (par exemple si vous avez tapé un "X"), le BASIC protestera en affichant

```
?REENTER
?
```

ou quelque message similaire. Ensuite il attendra votre nouvel essai.

L'Ordre IF

Lorsque vous avez entré un nombre, le BASIC le stocke dans le tiroir A et poursuit avec l'exécution de la ligne 150, qui dit :

```
IF A <> N1 + N2 THEN PRINT "DESOLE, C'EST FAUX
— RECOMMENCEZ" : GOTO 120
```

Il s'agit là d'un nouveau type d'instruction : l'instruction IF. En réalité, la phrase commençant par IF pouvant contenir plusieurs instructions, nous l'appellerons *ordre*. La première partie de l'ordre IF (ici, A <> N1 + N2) est appelée *condition*. (Le signe <> signifie « différent de ».) Si le nombre conservé dans le tiroir A n'est pas égal à

la somme des nombres contenus dans les tiroirs N1 et N2 (c'est le cas lorsque votre réponse est fausse), le BASIC exécutera la partie de l'ordre (appelé *action*) qui suit le mot THEN. L'action est elle-même constituée par une ou plusieurs instructions séparées les unes des autres par des points. Ici, les instructions sont

PRINT "DESOLE, C'EST FAUX — RECOMMENCEZ",

et

GOTO 120

Il est très important de comprendre le fonctionnement de l'ordre IF : si la condition est vraie, *toutes* les instructions qui la suivent sur la ligne sont exécutées ; en revanche, si la condition est fausse, *aucune* de ces instructions ne sera effectuée. Dans les deux cas, à moins qu'une instruction GOTO ne figure parmi celles qui suivent le mot THEN, le BASIC passe à la ligne suivante. Ainsi, à la figure 1 . 2, si la condition est vraie (c'est-à-dire si votre réponse est erronée), le BASIC exécutera d'abord l'instruction

PRINT "DESOLE, C'EST FAUX — RECOMMENCEZ"

puis l'instruction

GOTO 120

Nous savons déjà ce que fait l'instruction PRINT, aussi allons-nous étudier l'instruction GOTO, dont l'effet est de rompre l'ordre séquentiel de déroulement des instructions. Au lieu de passer à la ligne 160, le BASIC doit revenir à la ligne 120 et recommencer l'exécution à partir de ce point. Donc, si vous avez tapé un résultat inexact, la séquence « affichage question/attente réponse » sera répétée, ce qui donnera sur l'écran :

QUE FONT 5 + 7 ? **13**

DESOLE, C'EST FAUX — RECOMMENCEZ
QUE FONT 5 + 7 ?

(Votre réponse — 13 — est indiquée en caractères gras.)

Si la condition figurant dans l'ordre IF de la ligne 150 est fausse (ce qui est le cas, lorsque votre réponse est exacte), le reste de la ligne 150 est sauté et le BASIC passe à la ligne 160. Là, il exécute l'instruction

PRINT "EXACT — ESSAYEZ DONC ENCORE UNE FOIS"

puis, va à la ligne 170 où il exécute l'instruction

GOTO 100

Ce qui est affiché sur l'écran lorsque votre réponse est bonne, ressemble à ceci :

QUE FONT 5 + 7 ? **12**

C'EST EXACT — ESSAYEZ DONC ENCORE UNE FOIS
QUE FONT 9 + 2 ?

LE BASIC Libre

Comme nous l'avons déjà constaté, les instructions de la figure 1 . 2 suivent très fidèlement l'organigramme de la figure 1 . 1. La figure 1 . 3 montre le même programme en BASIC Libre. (Le BASIC Libre est décrit au chapitre 6.) Regardez la figure 1 . 3. Vous remarquerez que le programme en BASIC Libre ressemble beaucoup à celui de la figure 1 . 2; cependant, les lignes ne sont pas numérotées, et certains mots (comme « repeat » et « else ») qui figurent en minuscules sur la figure 1 . 3, n'existent pas sur la figure 1 . 2.

Le BASIC Libre est l'outil que l'auteur a utilisé pour concevoir les programmes BASIC qui figurent dans ce livre. Du chapitre 1 au chapitre 5, la description en BASIC Libre de chaque programme est fournie, mais les explications données ne se réfèrent jamais au BASIC Libre. Les descriptions en BASIC Libre y ont été ajoutées pour vous familiariser avec cette technique, et pour que les lecteurs,

```
repeat {
    N1 = INT(RND(1) * 10)
    N2 = INT(RND(1) * 10)
    repeat {
        PRINT " QUE FONT "; N1 ; " + "; N2
        INPUT A
        IF A = N1 + N2 THEN
            PRINT " EXACT etc."
        else
            PRINT " C'EST FAUX etc."
    } until (A = N1 + N2)
}
```

Voici la description en BASIC Libre du programme « Entraînement à l'Addition. Le BASIC Libre est expliqué au chapitre 6.

N.d.t. : Les mots en minuscules, tels « repeat » (répéter) « else » (dans le cas contraire), « until » (jusqu'à ce que) figureront dans leur forme anglaise dans les descriptions. En effet ces mots sont des mots réservés de certains langages évolués et/ou font partie du vocabulaire international de la programmation structurée.

Figure 1. 3 : L'Entraînement à l'Addition en BASIC Libre

qui reliraient ces chapitres après avoir pris connaissance du chapitre 6, puissent mieux comprendre de quelle façon les programmes présentés ont été réalisés. Les chapitres 7, 8 et 9 font appel à des descriptions en BASIC Libre uniquement, sans utiliser les instructions du BASIC réel.

L'Entraînement à l'Arithmétique

L'Entraînement à l'Addition n'était qu'un simple exercice de mise en train avant d'aborder le programme « Entraînement à l'Arithmétique ».

Le programme « Entraînement à l'Addition » nous a donné l'occasion d'étudier plusieurs importantes instructions BASIC. Mais en tant que jeu, il s'avère trop simple et trop monotone pour intéresser et stimuler le joueur. L'Entraînement à l'Arithmétique est un jeu similaire mais, apporte les deux caractéristiques supplémentaires suivantes :

— Les nombres peuvent avoir plus d'un chiffre (au gré du joueur).

— L'opération à effectuer, addition, soustraction, multiplication ou division est choisie au hasard.

Par ailleurs, l'Entraînement à l'Arithmétique ressemble beaucoup à l'Entraînement à l'Addition.

On trouvera ci-dessous, un exemple de dialogue entre le programme et le joueur. Notez que les réponses du joueur sont en caractères gras.

COMBIEN DE CHIFFRES ? **2**
QUE FONT 90/15 ? **6**

EXACT — ESSAYEZ DONC ENCORE UNE FOIS
QUE FONT 8 + 1 ? **9**

EXACT — ESSAYEZ DONC ENCORE UNE FOIS
QUE FONT 23 × 77 ? **1781**

DESOLE, C'EST FAUX — RECOMMENCEZ
QUE FONT 23 × 77 ?

Le Programme d'Entraînement à l'Arithmétique

L'organigramme est présenté figure 1.4. La figure 1.5 donne le listing du programme BASIC. Ce dernier est très semblable à celui de l'Entraînement à l'Addition de la figure 1.2. Plusieurs

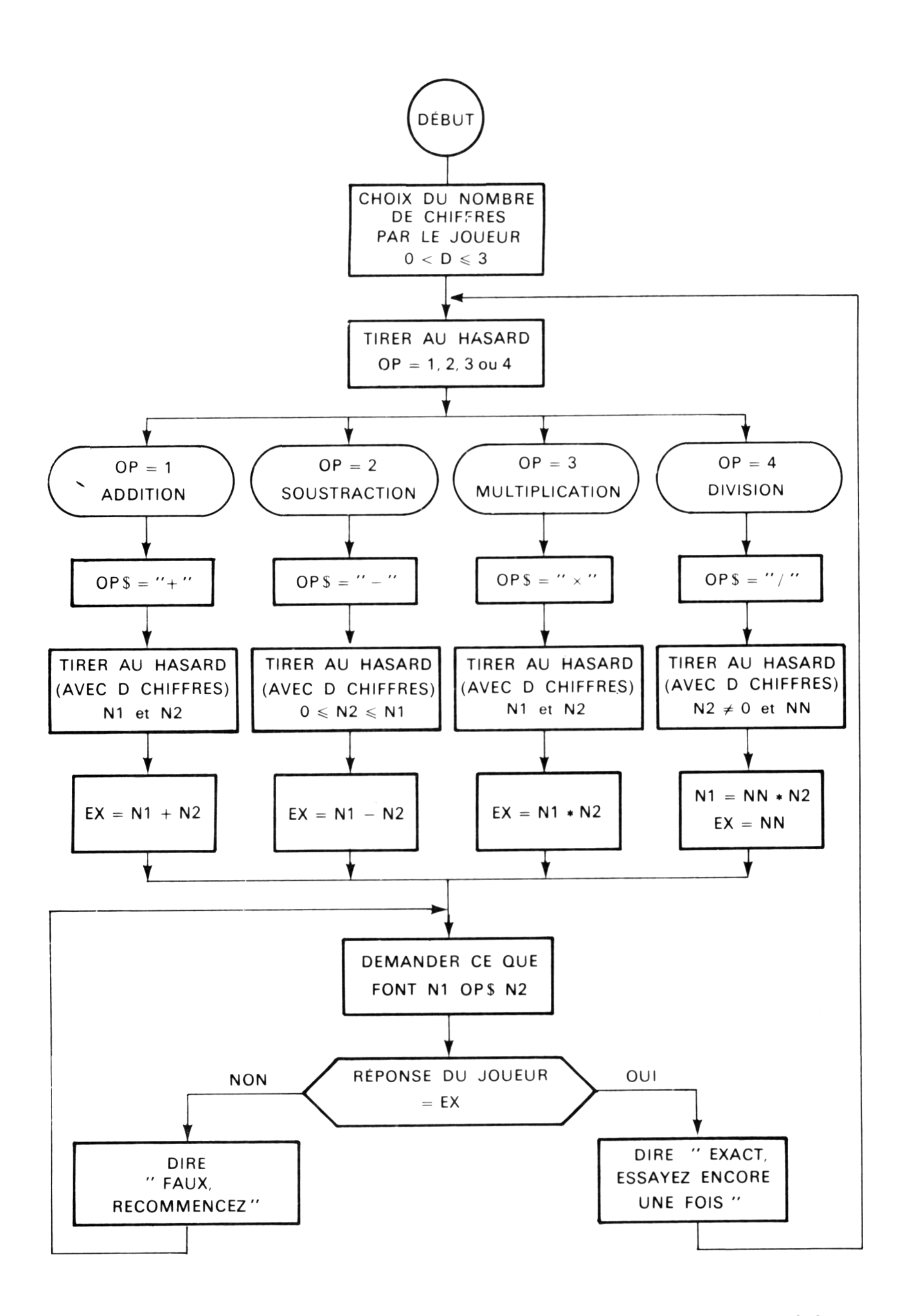


Figure 1.4 : Organigramme de l'Entraînement à l'Arithmétique

lignes de la figure 1.5 diffèrent légèrement de celles de la figure 1.2. Par exemple :

— La constante "+" à la ligne 130 de la figure 1.2 devient : OP$ à la ligne 150 de la figure 1.5.
— L'expression N1 + N2 à la ligne 150 de la figure 1.2 a été remplacée par la variable EX à la ligne 170.
— La formule INT(RND(1) * 10) aux lignes 100 et 110 de la figure 1.2 est devenue FNR(D) tout au long de la figure 1.5
ou FNR(D) = INT(RND(1) * 10↑D).

Ces changements sont appelés *généralisations*. La Généralisation consiste à remplacer un élément par un ensemble plus large dont cet élément est un cas particulier. Par exemple, le remplacement de "+" par OP$ est une généralisation parce que "+" n'est qu'une des quatre valeurs possibles de OP$ Il en est de même pour l'expression N1 + N2 par la variable EX, car N1 + N2 est une des quatre expressions qui permettent de calculer la valeur de EX. (Les autres expressions sont N1 − N2, N1 * N2 et N1/N2.) Enfin le remplacement de RND(1) * 10 par RND(1) * 10↑D est une généralisation parce que 10 (c'est-à-dire 10↑1), est une des trois valeurs que peut prendre 10↑D, en fonction des valeurs 1, 2 ou 3 possibles pour D.

La généralisation est une méthode qui a beaucoup d'importance pour le développement ultérieur des programmes. Chaque fois que cela est possible, les programmes doivent être conçus pour être facilement généralisables. Par exemple, nous pourrions avoir écrit l'Entraînement à l'Addition (fig. 1.2) de façon légèrement différente. Nous aurions pu ajouter en tête deux lignes :

```
80 OP$ = "+"
90 D = 1
```

ainsi qu'une autre ligne après la ligne 110 :

```
115 EX = N1 + N2
```

Les lignes 120 et 150 de la figure 1.2 auraient alors pu être identiques aux lignes 150 et 170 de la figure 1.5. L'avantage, est de mettre en évidence les éléments du programme qui peuvent facilement être généralisés. L'utilisation de variables au lieu de constantes (comme OP$ et D, à la place de "+" et 1) diminue le nombre des modifications qui seront nécessaires lorsque la partie correspondante du programme sera généralisée. Le risque d'oublier des modifications indispensables s'en trouve réduit.

```
100   INPUT "COMBIEN DE CHIFFRES";D
110   D=INT(D):IF D<=0 OR D>3 THEN 100
120*  DEF FNR(X)=INT(RND(1)*10↑X)
130*  OP=INT(RND(1)*4)+1
140   ON OP GOSUB 190,220,260,290
150   PRINT "QUE FONT";N1;OP$;N2;
160   INPUT A:PRINT
170   IF A<> EX THEN PRINT "DESOLE, C'EST FAUX—RECOMMENCEZ":GOTO 150
180   PRINT "EXACT—ESSAYEZ DONC ENCORE UNE FOIS":GOTO 130
190   OP$="+"
200   N1=FNR(D):N2=FNR(D)
210   EX=N1+N2:RETURN
220   OP$="-"
230   N1=FNR(D):NN=FNR(D):IF NN<=N1 THEN N2=NN:GOTO 250
240   N2=N1:N1=NN
250   EX=N1-N2:RETURN
260   OPS="X"
270   N1=FNR(D):N2=FNR(D)
280   EX=N1*N2:RETURN
290   OP$="/"
300   N2=FNR(D):IF N2=0 THEN 300
310   NN=FNR(D):N1=NN*N2
320   EX=NN:RETURN
```

Ce programme permet de jouer à l'Entraînement à l'Arithmétique. Le programme principal va de la ligne 100 à la ligne 180. Les quatre sous-programmes figurent aux lignes 190 à 210, 220 à 250, 260 à 280 et 290 à 320.

Les lignes 100 à 120 servent à préciser D (comme Digits, chiffres) c'est-à-dire le nombre de chiffres à l'intérieur des nombres proposés dans chaque exercice, et à définir la fonction FNR qui tire au hasard ces nombres.

Les lignes 130 et 140 choisissent l'une des quatre opérations (addition, soustraction, multiplication, division) comme sujet de l'exercice proposé et transfèrent l'exécution vers l'un ou l'autre des quatre sous-programmes de façon à préciser :

— la valeur des nombres N1 et N2 qui sont les données du problème
— la valeur du symbole OP$ placé entre eux (+, −, ×, /)
— la valeur du résultat escompté EX (comme Expected)

Les lignes 150 à 180 permettent de saisir la réponse du joueur, de vérifier son exactitude et suivant le cas de reposer la question ou de passer à l'exercice suivant.

Pour la raison déjà mentionnée à la figure 1 . 2 et parce qu'en outre sur l'Apple, le BASIC ne met pas de point d'interrogation à la fin des chaînes de caractères incluses dans l'instruction INPUT le programme ci-dessus n'est valable que sur PET et TRS-80. Sur Apple les lignes 100 et 150 doivent devenir :

```
100   INPUT "COMBIEN DE CHIFFRES";D
150   PRINT "QUE FONT"; N1; OP$; N2;"";
```

* Sur TRS-80 on doit remplacer RND(0) par RND(1) (voir figure 1 . 2).

Figure 1.5 : L'Entraînement à l'Arithmétique

L'utilisation de variables à la place de constantes et la prévision des généralisations éventuelles sont d'excellentes habitudes de programmation. Cela n'a pas été fait dans le cas de la figure 1.2 parce que cela aurait augmenté la complexité des explications fournies pour ce premier programme élémentaire.

Et maintenant, regardons l'organigramme de la figure 1.4. Remarquons qu'il est identique à celui de la figure 1.1 sauf sur deux points :

— La case contenant « Choix par le joueur... » ne se trouve pas dans la figure 1.1.

— Les mots « Choisir deux nombres de 1 chiffre, N1 et N2 » ont été remplacés dans la figure 1.4 par « Tirage au hasard de OP = 1, 2, 3 ou 4 ». La case contenant ces mots est suivie par quatre chemins différents dont un seul est parcouru à chaque itération.

Ces deux points concernent l'affectation de valeurs aux variables D, OP$ et EX. En conséquence, les parties de l'organigramme de la figure 1.4 qui traitent des généralisations, développent les parties correspondantes de la figure 1.1 ; par contre, les éléments qui ne sont pas concernés par les généralisations, restent identiques d'une figure à l'autre.

Et maintenant regardons le programme de la figure 1.5. Il illustre des particularités du BASIC que nous n'avons pas encore mentionnées. Les deux premières lignes ont pour objet l'affectation d'une valeur à la variable D (D symbolise DIGIT qui signifie Chiffre.) La première de ces lignes est :

```
INPUT "COMBIEN DE CHIFFRES"; D
```

La constante "COMBIEN DE CHIFFRES" précise la question que le programme BASIC doit poser; la variable D indique au BASIC où stocker la réponse. En conséquence, cette instruction amène le BASIC à demander :

```
COMBIEN DE CHIFFRES ?
```

Le BASIC attend la réponse (numérique) puis stocke la valeur dans le tiroir D. Ceci n'est pas une règle générale. La syntaxe BASIC de la figure 1.5 est celle du PET et du TRS-80. Sur Apple il faudrait pour arriver à un résultat identique écrire :

```
INPUT "COMBIEN DE CHIFFRES ?"; D
```

Sur PET et TRS-80, le BASIC ajoute un « ? » à la fin de la constante figurant dans le corps de l'instruction, ce n'est pas le cas sur Apple.

La ligne suivante contient deux instructions séparées par deux points :

```
D = INT(D) : IF D < = 0 OR D > 3 THEN 100
```

Nous avons déjà appris dans ce chapitre que la partie « action » d'un ordre IF peut être composée de plusieurs instructions séparées les unes des autres par deux points. (Mais rappelez-vous que les instructions à exécuter à la suite d'un « IF » sont toutes situées dans la partie « action » de l'ordre IF.) Nous connaissons déjà INT, qui transforme D en nombre entier, et permet donc de se « débarrasser » des décimales. Dans l'ordre IF, le nombre 100 qui suit le mot THEN est l'abréviation de l'instruction :

```
GOTO 100
```

La ligne 120 du programme contient une *définition de fonction* :

```
DEF FNR(X) = INT(RND(1) * 10↑X)
```

Une définition de fonction est un moyen d'indiquer au BASIC qu'on désire abréger une formule trop longue. L'instruction ci-dessus précise que FNR(D) sera l'abréviation de l'expression INT(RND(1) * 10↑D). Cette abréviation est utilisée aux lignes 200, 230, 270, 300 et 310.

La variable X qui figure des deux côtés du signe égal dans la définition de fonction porte le nom de variable « factice » (on dit aussi « muette ». Une variable « muette » sert à indiquer la place où l'argument réel (D en l'occurrence) devra être utilisé dans la formule, chaque fois qu'on voudra en calculer la valeur. Par exemple, si nous écrivons FNR(2), un 2 sera utilisé à tous les endroits où figure X, de sorte que FNR(2) donnera le même résultat que INT(RND(1) * 10↑2).

La ligne suivante de notre programme, c'est-à-dire la ligne 130, contient l'instruction :

```
OP = INT(RND(1) * 4) + 1
```

Elle entraîne le choix au hasard de l'un des nombres 1, 2, 3 ou 4 et son stockage dans le tiroir OP. (Si vous ne comprenez pas clairement pourquoi, vous pouvez relire les explications données précédemment sur les raisons qui font que INT(RND(1) * 10) choisit au hasard une valeur entière comprise entre 0 et 9.

Notons, au passage, que chaque fois que nous avons associé les fonctions INT et RND, c'était pour tirer au hasard un nombre dans une suite de nombres entiers consécutifs (0 à 9, 0 à 99, 0 à 999, 1 à

4). Si BASIC était un peu plus souple d'emploi, nous aurions pu écrire, pour traiter l'ensemble de ces cas, la définition suivante :

DEF FNC(L,H) = INT(RND(1) * (H − L + 1)) + L

[*Nota* : La variable H est l'abréviation de HIGH (haut) et la variable L est celle de LOW (bas). Il s'agit des bornes hautes et basses de l'intervalle de choix. *N.d.T.*]

Ainsi,

OP = FNC(1,4)

tirerait un nombre entier compris entre 1 et 4, alors que

N1 = FNC(0,9)

tirerait un nombre entre 0 et 9. L'expression FNR(D) serait alors remplacée par FNC(0,10↑D − 1). Malheureusement ce genre de définition de fonction n'est pas autorisé en BASIC standard qui ne permet pas les définitions de fonction comportant deux variables muettes. En conséquence, nous devrons écrire l'expression complète INT(RND(1) * (H − L + 1)) + L partout où nous aurions voulu utiliser FNC(L,H).

Si nous revenons au programme de la figure 1.5, nous voyons que la ligne suivante numérotée 140 nous propose quelque chose que nous n'avons pas étudié : l'ordre ON :

ON OP GOSUB 190, 220, 260, 290

Rappelons que nous venons d'affecter à la variable OP l'une des valeurs 1, 2, 3 ou 4. Il y a 4 numéros de lignes (190, 220, 260 et 290) cités dans l'instruction. Ces lignes sont le début de quatre sous-programmes (Subroutines). Si OP vaut 1, le sous-programme commençant à la ligne 190 sera alors exécuté ; si OP vaut 2, ce sera le sous-programme de la ligne 220 ; si la valeur est 3, ce sera la ligne 260 et enfin la ligne 290 si celle-ci est 4. Une seule des routines est « appelée ». Ce qui se produit si OP prend une valeur n'appartenant pas à l'ensemble des nombres 1, 2, 3 et 4, diffère selon les systèmes.

OP et les quatre sous-programmes ont pour objet de représenter les quatre branches prévues dans l'organigramme de la figure 1.4. Chacun des quatre sous-programmes correspond à l'une des quatre opérations de l'Entraînement à l'Arithmétique : l'addition, la soustraction, la multiplication et la division. Chaque sous-programme affecte des valeurs aux variables NI, OP$, N2 et EX ; c'est-à-dire au premier nombre, au symbole de l'opération, au deuxième nombre et à la réponse attendue (Expected). Nous verrons le détail des sous-

programmes lorsque nous aurons terminé l'analyse du programme principal. La ligne suivante, numérotée 150, contient l'instruction :

```
PRINT "QUE FONT"; N1; OP$; N2;
```

Cette ligne ressemble beaucoup à la ligne 120 de la figure 1.2, mais dans le programme de la figure 1.5, la variable OP$ a remplacé la constante « + ». Les deux instructions de la ligne 160 :

```
INPUT A: PRINT
```

sont identiques à celles des lignes 130 et 140 de la figure 1.2. L'instruction INPUT fait entrer en mémoire la réponse du joueur. L'Instruction PRINT ménage un interligne pour améliorer la lisibilité.

Les lignes 170 et 180 correspondent aux lignes 150, 160 et 170 de la figure 1.2. Les seules différences concernent l'apparition de EX (à la place de N1 + N2) et la numérotation des lignes. Nous en avons terminé avec le programme principal.

Les quatre sous-programmes se ressemblent beaucoup. Le premier (lignes 190, 200, 210) est le plus simple :

```
OPS = "+"
N1 = FNR(D) : N2 = FNR(D)
EX = N1 + N2 : RETURN
```

Ce sous-programme montre quelle est la structure générale de chacune des quatre routines. Une valeur est d'abord affectée à OP$. Dans le cas présent, la valeur que prend OPS est "+", signe de l'addition. Puis, deux nombres sont tirés et stockés dans leurs « tiroirs » respectifs N1 et N2. Enfin, le résultat attendu est mémorisé dans la variable EX.

Le deuxième sous-programme (lignes 220, 230, 240 et 250) est un peu plus compliqué :

```
    OP$ = "-"
    N1 = FNR(D) : NN = FNR(D) :
    IF NN < = N1 THEN N2 = NN : GOTO 250
    N2 = N1 : N1 = NN
250 EX = N1 - N2 : RETURN
```

Sa complexité tient au fait que nous désirons que le premier nombre soit supérieur ou égal au second de façon à éviter les résultats négatifs. On commence par déterminer N1 puis, le deuxième nombre est placé dans NN. Si ce second nombre est inférieur ou égal au premier, il est tout simplement recopié en N2. Mais si le nombre en N1 est plus petit que celui en NN, les deux nombres sont permutés.

Celui qui est en N1 doit être recopié en N2, et celui qui est en NN est alors recopié en N1. Une fois que les valeurs correctes ont été ainsi attribuées à N1 et N2, le résultat attendu est la différence de ces deux nombres.

La troisième routine (lignes 260, 270 et 280) est presque identique à la première :

```
OP$ = "X"
N1 = FNR(D) : N2 = FNR(D)
EX = N1 * N2 : RETURN
```

La seule différence réside dans le remplacement des symboles de l'addition par ceux de la multiplication.

Le quatrième sous-programme (lignes 290, 300, 310 et 320), est le plus complexe car il doit remplir deux conditions :

— Le diviseur ne doit pas être égal à zéro.

— Le résultat doit être un nombre entier.

Il est facile de satisfaire ces deux exigences :

```
    OP$ = "/"
300 N2 = FNR(D) : IF N2 = 0 THEN 300
    NN = FNR(D) : N1 = NN * N2
    EX = NN : RETURN
```

La ligne 300 recommence le tirage du diviseur jusqu'à ce que la valeur obtenue soit différente de zéro. Ensuite, au lieu de déterminer le premier nombre, nous tirons au hasard le résultat. Le premier nombre est alors calculé en multipliant le diviseur par le résultat.

Ceci termine notre analyse du programme « Entraînement à l'Arithmétique ». La figure 1.6 présente une description en BASIC Libre. Comme nous l'avons déjà précisé, vous n'êtes pas obligé de vous en servir si cela ne vous intéresse pas. Si néanmoins vous l'étudiez, vous pourrez remarquer plusieurs petites différences entre cette description de la figure 1.6 et le programme de la figure 1.5.

Additions et modifications possibles

Pour terminer, à l'intention des lecteurs qui désireraient améliorer le jeu, voici quelques suggestions. Certaines sont faciles, d'autres plus difficiles à programmer.

- Etablir des statistiques : par exemple enregistrer le nombre de réponses correctes données depuis la dernière réponse fausse,

et le plus grand nombre de réponses correctes consécutives antérieurement obtenues. Afficher ces nombres après chaque réponse correcte.

- Mémoriser les exercices ayant donné lieu à une réponse fausse et les proposer au joueur de temps à autre.

```
# Entraînement à l'Arithmétique
        DEF  fnc(L,H)=INT(RND(1) * (H-L+1))+L    # fonction de tirage au
                                                  hasard
        repeat
             INPUT "COMBIEN DE CHIFFRES"; D
             until (D = 1 OR D = 2 OR D = 3)
        L = 0 : U = 10↑D - 1                      # Les limites des nombres
                                                  sont fixées

        repeat {                                  # Boucle principale
             OP = fnc(1,4)                        # Tirage d'un nombre entre
                                                  1 et 4
             ON OP GOSUB add, soust, mul, div
             repeat {                             # répéter la question jus-
                                                  qu'à obtention de la réponse
                                                  correcte
                  PRINT "QUE FONT"; N1 ; OP$; N2;
                  INPUT A : PRINT
                  IF A <> EX THEN PRINT " C'EST FAUX
                  ect "
                  } until (A = EX)
             PRINT "EXACT ect "
             }
add     OP$ = " + "
        N1 = fnc(L, U) : N2 = fnc(L, U)           # Tirer deux nombres
        EX = N1 + N2 : RETURN                     # Prévoir la somme

soust   OP$ = " - "
        N1 = fnc(L, U) : N2 = fnc(L, U)           # Tirer deux nombres
        IF N1 < N2 THEN swap N1, N2               # Mettre le plus grand en
                                                  première position
        EX = N1 - N2 : RETURN                     # Prévoir la différence

mul     OP$ = " × "
        N1 = fnc(L, U) : N2 = fnc(L, U)           # Tirer deux nombres
        EX = N1 * N2 : RETURN                     # Prévoir le produit

div     OP$ = "/ "
        repeat N2 = fnc(L, U) until (N2 <> 0)     # Tirer le diviseur
        EX = fnc(L, U)                            # Tirer au hasard le quotient
        N1 = EX * N2 : RETURN                     # Calcul du dividende
```

Figure 1.6 : L'Entraînement à l'Arithmétique en BASIC Libre

- Ajouter d'autres opérations comme l'élévation au carré ou au cube, les puissances de 2 et le passage de base 2 en base 10.
- Contrôler la position du curseur pour rendre stable l'affichage : on peut, par exemple, afficher la question posée toujours au même endroit de l'écran. Il en est de même pour le résultat du coup et le score. On pourrait même faire disparaître la question précédente et son résultat quelque temps avant d'afficher la nouvelle question.
- Donner la possibilité au joueur de choisir les opérations arithmétiques auxquelles il désire s'entraîner.
- Incorporer au jeu le facteur temps (sur les systèmes ayant une horloge incorporée comme le Pet) en imposant au joueur un délai de réponse ou en mémorisant le temps de réponse moyen (ou médian) pour chaque type d'exercice. Si le temps de réponse est limité, afficher le compte à rebours des secondes restantes (voir chapitre 3).
- Sonoriser le jeu (sur les systèmes munis de haut-parleurs comme l'Apple) pour saluer par des applaudissements les réponses exactes et par des quolibets les réponses fausses. Rendre la sonorisation facultative de façon à ce que le joueur puisse la neutraliser de manière simple à partir du clavier.

En résumé

Nous venons d'étudier un jeu facile : l'Entraînement à l'Addition, et un autre plus complexe : l'Entraînement à l'Arithmétique. En analysant l'Entraînement à l'Addition, nous avons fait connaissance avec la génération de nombres au hasard (utilisation de INT et de RND), ainsi qu'avec l'instruction PRINT, les variables, les constantes, l'instruction INPUT et l'ordre IF.

Le jeu « Entraînement à l'Arithmétique » et son programme ont illustré le concept de généralisation et l'utilisation de variables à la place de constantes (technique majeure de généralisation ultérieure des programmes). Nous avons également vu les constantes du type chaîne de caractères intégrées aux instructions « INPUT », ainsi que les définitions de fonctions et leurs variables muettes, la structure ON...GOSUB, et l'utilisation des sous-programmes (SUBroutines).

Nous avons analysé rapidement les descriptions en BASIC Libre, et l'utilisation de ce langage a été expliquée. Les descriptions en BASIC libre sont données pour tous les programmes mais les cinq premiers chapitres peuvent être lus sans avoir à s'y reporter.

En dernier lieu, nous avons proposé des additions et des améliorations au jeu de l'Entraînement à l'Arithmétique ; statistiques, opérations supplémentaires, choix des opérations par le joueur ainsi que l'utilisation de dispositifs particuliers comme l'horloge et le haut-parleur.

CHAPITRE **2**

Jeux de la Découverte

Dans le chapitre 1, nous avons choisi de façon délibérée des jeux simples. L'analyse des instructions BASIC qui sont utilisées, a mis l'accent beaucoup plus sur la syntaxe du BASIC que sur les méthodes de programmation et de conception des jeux. Cependant nous avons étudié dans le chapitre 1, un concept important : la généralisation. Ce concept jouera également un rôle essentiel dans les jeux de ce chapitre.

Nous allons présenter maintenant plusieurs jeux qui consistent à découvrir un mot ou nombre inconnu. Notre étude commencera par un rappel de la structure générale des Jeux de la Découverte. Puis nous détaillerons les règles du jeu appelé « Quatre » et donnerons à titre d'exemple, une suite de coups tentés par le joueur et les répliques correspondantes de l'ordinateur. Cet exemple montrera la stratégie suivie par un bon joueur. Comprendre l'esprit des jeux vous aidera à concevoir des jeux intéressants et stimulants.

Plutôt que de vous fournir un programme qui ne soit joué qu'à « Quatre », nous prévoirons d'abord la généralisation. Pour cela, nous présenterons une famille de jeux ayant pour thème la découverte d'un nombre. Ces jeux sont appelés Un, Deux... jusqu'à Neuf (Quatre n'étant qu'un des membres de la famille). Puis, pour généraliser encore plus, nous présenterons un jeu de découverte d'un mot. Ce jeu appelé le « Mot » a des règles proches de celles des jeux de nombres.

Ce n'est qu'à ce moment-là que nous construirons notre programme. Dans ce programme dit du « Jeu de la Découverte », vous trouverez de nombreux principes et méthodes de programmation très pratiques. L'une de ces techniques étant la « cannibalisation » d'un programme pour en tirer un programme similaire, nous présenterons d'abord, le jeu du « Pendu » et terminerons le chapitre en « cannibalisant » le « Jeu de la Découverte » pour en faire le programme du Pendu.

Structure générale des Jeux de la Découverte

Les jeux du type que nous allons décrire tendent à se ressembler. En général, les joueurs sont au nombre de deux. Le premier pense à un nombre ou à un mot. Le second joueur fait un certain nombre d'essais. Les indications que donne le premier joueur en réponse à ces essais, ont pour but de réduire le champ des nombres ou des mots possibles, de telle sorte que tôt ou tard, le deuxième joueur peut proposer sa

solution en toute certitude. Qui plus est, le nombre d'essais autorisé est généralement limité.

Quand ces jeux se jouent sur ordinateur, tout ou partie du rôle du premier joueur est tenu par le programme. En conséquence, le deuxième joueur peut faire confiance à l'exactitude des commentaires faits par le premier joueur en réponse aux propositions de solution. Quand nous examinerons, à titre d'exemple, une suite de coups lors d'une partie de Quatre, nous verrons comment le second joueur utilise au maximum l'information contenue dans les réponses du premier. Si le premier joueur (le programme ici) répond de façon inexacte, il va de soi que ce jeu de réflexion devient un jeu de hasard.

Quatre

Quatre est un jeu simple dont il existe de nombreuses variantes. Le programmeur joue en premier en choisissant un nombre « secret » de 4 chiffres, ces chiffres étant différents les uns des autres. C'est alors à vous de jouer. C'est-à-dire que vous devez découvrir non seulement les quatre chiffres mais aussi l'ordre dans lequel ils se trouvent. Par exemple, si le nombre caché est 7915, il nous faudra proposer 7915 ; 1975 n'étant pas suffisant.

Après chaque tentative, le programmeur répond par deux nombres. Le premier correspond aux chiffres de votre proposition effectivement présents dans le nombre à découvrir. Par exemple, si le nombre à découvrir est 7915 et votre essai 1975, le premier nombre indiqué en réponse par le programme sera 4, puisque les quatre chiffres de 1975 figurent dans 7915. Le deuxième nombre donné par le programmeur indique combien de chiffres sont à leur place exacte. Par exemple en reprenant les mêmes hypothèses, le deuxième nombre indiqué par le programme sera 2, car 9 et 5 sont à leur place. Ce dialogue figurerait sur l'écran à peu près comme ceci :

QUE PROPOSEZ-VOUS ? **1975**
EXACTS : 4 EN PLACE : 2
QUE PROPOSEZ-VOUS ?

Cette suite de propositions et de commentaires se poursuit jusqu'à ce que vous trouviez la solution. Si vous épuisez le nombre de coups autorisé avant de trouver, le programmeur mettra fin à la partie de sa propre autorité.

Un exemple de partie

La figure 2.1 montre un exemple de partie. Examinons ce dialogue ligne par ligne pour nous assurer que nous comprenons bien le dérou-

le programme tire le nombre 7359.

3 est un chiffre de 7359, mais situé à la seconde place, pas à la troisième.

5 et 7 sont des chiffres de 7359, mais en troisième et première places, pas en première et troisième.

9 est un chiffre de 7359 mais se trouve à la dernière place et non à la première.

3, 7 et 9 sont des chiffres de 7359. 9 est à la dernière place comme prévu.

3 et 9 sont des chiffres de 7359, mais en seconde et quatrième positions et non pas en quatrième et troisième.

3, 9, 5 et 7 sont tous des chiffres de 7359 mais seul 5 est à sa place.

Le programmeur tire au hasard un nombre de quatre chiffres. Le joueur propose des solutions et le degré d'exactitude de chaque proposition est indiqué. Comme on le voit ci-dessus, le joueur a préparé chaque coup pour en retirer le plus de renseignements possible et pour utiliser au mieux les informations retirées des coups précédents.

Tout joueur expérimenté jouera au moins aussi bien que dans l'exemple ci-dessus : un coup malheureux, le quatrième, en raison d'un chiffre à sa place, a restreint le nombre de renseignements disponibles au moment du sixième coup. Si le quatrième coup avait été 3967, le joueur aurait connu non seulement les chiffres, mais également la position occupée par ceux-ci au moment du sixième coup.

Figure 2.1 : Une partie de « Quatre »

lement du jeu. Nous commenterons également les coups du joueur pour que vous ayez une idée des « astuces » que vous pouvez utiliser pour trouver la solution rapidement.

Le premier essai du joueur est fait avec un seul renseignement ; le programmeur vient de choisir un nombre de 4 chiffres dont tous les chiffres sont différents. Le choix par le joueur du nombre 1234 n'est ni meilleur ni pire que n'importe quel autre. La réponse du programmeur : "EXACTS : 1, EN PLACE : 0" indique qu'un seul des chiffres de la solution figure parmi les chiffres 1, 2, 3 et 4 et qu'aucun des chiffres exacts de l'essai ne se trouve à sa place. Ce dernier renseignement s'avérera très utile dans la suite de la partie.

Le second essai, 5678, contient quatre nouveaux chiffres. Notons que dans les premiers coups, le joueur cherche à essayer la totalité des dix chiffres possibles. Cela l'aide à identifier les quatre chiffres très rapidement. La réponse "EXACTS : 2, EN PLACE : 0" indique au joueur que deux des chiffres de la solution font partie de l'ensemble 5, 6, 7, 8. Le joueur peut maintenant en déduire que le quatrième chiffre est soit 9, soit 0 puisque le total des chiffres exacts n'est que de trois et que 9 et 0 sont les seuls chiffres à ne pas avoir été essayés.

Le troisième essai, 9012, a pour but de savoir ce qu'il en est des chiffres 9 et 0 et de commencer à préciser quel chiffre parmi 1, 2, 3 ou 4 fait partie de la solution. La réponse "EXACTS : 1, EN PLACE : 0" indique au joueur que les chiffres 1 et 2 sont éliminés. EXACTS : 1 ne peut correspondre qu'à 9 ou à 0 (comme nous l'avons déjà vu). Donc les chiffres 1 et 2 sont faux. Le joueur sait maintenant qu'un des chiffres est soit 3, soit 4, un autre étant soit 9, soit 0 et les deux qui restent, étant à prendre parmi 5, 6, 7 et 8.

Le quatrième essai, 3769, est formé de l'un des deux chiffres 3 et 4, de deux autres chiffres pris dans 5, 6, 7 et 8 et d'un dernier, choisi entre 9 et 0. La réplique "EXACTS : 3, EN PLACE : 1" fournit au joueur des indications très utiles sur les chiffres composant le nombre. L'ironie du sort fait qu'on aurait eu davantage de renseignements sur la place des chiffres, si au lieu d'en avoir un à sa place, on n'en avait pas eu du tout. Nous verrons pourquoi, lorsque nous étudierons quel coup il faut jouer au sixième essai.

Le cinquième essai, 6893, a été décidé à la suite du raisonnement suivant : après le quatrième essai, on savait que si 3 et 9 étaient corrects, 6 et 7 ne pouvaient l'être simultanément et vice versa. (Il ne pouvait pas y avoir un chiffre inexact dans chacune de ces paires puisque le total des chiffres erronnés n'était que de un.) Etant donné que 3 et 9 avaient chacun 50 chances sur 100 d'être exacts (car l'un des chiffres est soit 3 soit 4, l'autre étant 0 ou 9), la probabilité pour

que les deux chiffres corrects soient simultanément 3 et 9, est de 1 sur 4. Parallèlement, comme il y a six façons de combiner deux à deux, quatre chiffres différents, la probabilité que la paire 6, 7 soit correcte est de 1 sur 6. En conséquence, le joueur a décidé de supposer que 3 et 9 font partie du nombre secret et qu'un seul des deux chiffres 6 et 7 y est. La réplique "EXACTS : 2, EN PLACE : 0" apprend au joueur que le 7 est à coup sûr dans le nombre, puisque le seul changement (hormis la position occupée) entre le quatrième et le cinquième essai, a consisté à remplacer 7 par 8. Cette réponse indique aussi au joueur que le 8 est éliminé de façon certaine.

Pour son sixième essai, 3957, le joueur a décidé de s'en tenir à l'hypothèse d'exactitude du 3 et du 9. Si cette hypothèse est exacte, 6 doit être faux puisqu'alors les trois chiffres corrects du quatrième essai (3769) auraient été 3, 9 et 7. Cette élimination du 6 et du 8 laisse 5 comme autre chiffre possible, puisque deux chiffres sont à prendre parmi 5, 6, 7 et 8.

La place des chiffres dans le sixième essai s'appuie sur l'examen des renseignements qu'ont donné à ce sujet les coups précédents. Notons que lors des cinq premiers essais, aucun chiffre ne se trouve à la même place dans deux essais différents. Etant donné que quatre de ces essais ont reçu comme appréciation "EN PLACE : 0", le joueur peut éliminer certaines positions pour chacun des quatre chiffres retenus. Il ne reste que les possibilités suivantes :

3 peut être seulement en première ou en seconde position.
5 peut se trouver en seconde, troisième ou quatrième position.
7 peut être en première, seconde ou quatrième position.
9 peut se trouver en seconde ou quatrième position.

A partir de là, on s'aperçoit que le 5 ne peut se trouver qu'en troisième position. Les certitudes ne pouvant aller plus loin, le joueur a décidé d'essayer le 9 en deuxième position. Une fois cette décision prise, le 3 ne peut être placé qu'à la place du numéro un. Seule la place quatre reste disponible pour le 7.

Le jugement "EXACTS : 4, EN PLACE : 1" prouve que les quatre chiffres sont corrects, mais que leur position ne l'est pas.

Le coup final, 7359, est joué à coup sûr. Le joueur sait déjà que le 5 est à sa place. Il sait que le 9 ne peut pas être en deuxième position. Le 9 ne peut donc se situer qu'en quatrième. La dernière décision est à prendre entre 3759 et 7359. Mais au quatrième coup (3769) il n'y avait qu'un seul chiffre en place. Il convient donc d'éliminer la solution 3759. Le nombre ne peut être que 7359. Nous voici arrivés à la fin de notre exemple de partie.

Notre explication du jeu de Quatre mettra l'accent sur trois points :

1. Comment rassembler plusieurs jeux analogues en un même programme.
2. Comment « cannibaliser » un programme existant pour créer un nouveau programme le plus rapidement possible.
3. Comment tirer parti d'un certain nombre de recettes de programmation.

Quelques jeux analogues à Quatre

Pour illustrer le premier point, il nous faut trouver quelques jeux voisins de Quatre. Deux cas évidents sont les jeux Trois et Cinq. Ces jeux se jouent comme Quatre, mais avec des nombres de trois et cinq chiffres. Le programme du Jeu de la Découverte que nous analyserons bientôt, permet de jouer avec des nombres allant de un jusqu'à neuf chiffres.

La généralisation qui permet d'incorporer au programme de Quatre les jeux de Trois et Cinq nécessite une préparation bien qu'elle soit, dans son principe, très simple. Pour rendre la chose un peu plus compliquée, le Jeu de la Découverte devra permettre de jouer également au Mot. Le Mot se joue comme Quatre, mais avec trois différences importantes :

— Le joueur essaie de trouver un mot et non un nombre.
— Une lettre donnée peut figurer plus d'une fois dans le mot (dans les jeux portant sur des nombres, les chiffres doivent être différents).
— Le mot à trouver est introduit en mémoire par un deuxième joueur (pour les jeux de nombres c'est l'ordinateur qui composent ces derniers de façon aléatoire).

Ces différences vont nous obliger à analyser de façon détaillée notre programme du Jeu de la Découverte. Quand la conception de ce programme sera terminée, il devrait être possible d'y incorporer d'autres jeux et ceci sans programmation compliquée. Des suggestions concernant ces additions sont données en fin de chapitre.

Le Pendu

En ce qui concerne le second point — comment réutiliser (ou « cannibaliser ») un programme existant — nous utiliserons le jeu dit du Pendu. Le Pendu ressemble aux jeux déjà décrits, mais comme vous allez le voir, il en diffère aussi sur des points importants.

L'un des joueurs choisit un mot et précise le nombre d'erreurs autorisé. L'autre joueur essaie alors de trouver le mot. Le programme commence par afficher une série de tirets correspondant au nombre de lettres du mot. Si le mot est, par exemple, SYZYGY et si le nombre d'erreurs permis est de cinq, l'affichage suivant apparaîtra :

— — — — — — VOTRE CREDIT D'ERREURS : 5
QUE PROPOSEZ-VOUS ?

Le joueur doit alors, soit proposer la totalité du mot, soit essayer une seule lettre. Si le joueur répond par une lettre unique et que celle-ci soit dans le mot, les tirets correspondants sont remplacés par la lettre en question. Par exemple, si le joueur tente Y, on verra apparaître :

— Y — Y — Y VOTRE CREDIT D'ERREURS : 5
QUE PROPOSEZ-VOUS ?

La partie se poursuit jusqu'à ce que le joueur trouve le mot ou dépasse le nombre d'erreurs permis.

Il est certain que notre Jeu de la Découverte pourrait être conçu de telle façon que le jeu du Pendu y soit inclus. Néanmoins l'étendue des différences existant entre le Pendu et les autres jeux est telle qu'il est nettement plus facile de faire un programme à part, pour le Pendu. Nous obtiendrons ce programme en modifiant ou en remplaçant certaines parties du programme de notre Jeu de la Découverte.

Recettes de programmation

Nous voulons en troisième lieu attirer l'attention sur quelques « astuces » de programmation utilisées. La plus importante concerne le codage du mot ou du nombre à découvrir ainsi que le procédé utilisé pour comparer le coup joué à cette solution codée. La technique qui consiste à condenser plusieurs informations à l'intérieur d'un même nombre en additionnant des puissances de deux, est semblable à celle qu'on utilise en langage machine où les bits d'un même octet ont des fonctions différentes. Nous verrons en détail cette technique quand nous étudierons la table « coups » utilisée pour coder le nombre ou le mot « secret ».

Et maintenant que nous avons vu quels sont les jeux qui feront partie de notre programme, que nous avons dévoilé notre intention de le « cannibaliser » au profit du Pendu et que nous avons montré quelles sont les techniques mises en œuvre, nous allons nous pencher sur le programme lui-même.

Le programme du Jeu de la Découverte

Les figures 2.2 à 2.12 présentent ce programme. Le « sous-programme principal » est montré à la figure 2.2. Notons qu'il apparaît deux fois. Dans le haut de la figure se trouve la description en BASIC Libre et dans le bas, les instructions en BASIC correspondantes. Comme nous l'avons précisé antérieurement, le BASIC Libre ne se trouve là que pour vous devenir familier, ou vous aider si vous avez d'abord lu le chapitre 6. Vous pouvez ne pas vous en occuper ; mais si vous comparez les deux formes (BASIC Libre et BASIC) données pour ce programme (ainsi que pour les suivants) vous parviendrez à comprendre le BASIC Libre même si vous n'avez pas lu le chapitre 6.

Le programme du Jeu de la Découverte de la figure 2.2 est constitué par 3 boucles imbriquées. La boucle intérieure (lignes 140 et 150) correspond aux solutions proposées par le joueur pour le mot ou le nombre à découvrir. La boucle est répétée jusqu'à ce que le joueur ait trouvé toutes les lettres (ou tous les chiffres) et les ait placées dans l'ordre (condition IP = N), ou jusqu'à ce que le crédit de coups permis ait été épuisé (condition IP = − 1). A l'intérieur de cette boucle, les sous-programmes des lignes 360, 410 et 480 sont appelés pour saisir la proposition du joueur, la comparer à la solution cachée et indiquer combien de chiffres ou de lettres sont « EXACTS » et combien sont « EN PLACE ». La variable IP (in place, en place) qui contrôle la boucle est calculée par la routine de la ligne 410.

La boucle de niveau immédiatement supérieur correspond à tout le reste du programme, mis à part les lignes 100 et 110. Cette boucle permet de jouer une suite de parties pour un type de jeu donné. A la fin de chaque partie, on doit indiquer au programme, en frappant une seule touche, ce qu'il faut faire ensuite. Le caractère tapé par le joueur est saisi par le sous-programme de la ligne 710 (appelée depuis la ligne 160) et stocké dans la variable NX$. Les instructions de la boucle sont exécutées à nouveau (par exemple, si l'on joue une nouvelle partie du même jeu) jusqu'au moment où NX$ (qui contient le caractère frappé par le joueur) prend la valeur " N " ou " E ". " N " signifie que le joueur désire passer à un nouveau jeu. " E " signifie que le joueur ne veut plus jouer.

La boucle extérieure comprend la ligne 110, laquelle appelle un sous-programme qui invite le joueur à choisir le type de jeu qui lui plaît. Au début de chaque exécution et à chaque fois que le joueur tape " N " à la fin de la partie, le programme vient ou revient à la ligne 110 pour permettre de choisir le type de jeu.

La ligne 100 qui comporte l'appel à une routine « d'initialisation » située en 670, n'est exécutée qu'une seule fois en début de programme. Elle ne fait partie d'aucune boucle et ne sera jamais exécutée une deuxième fois. De façon analogue, l'instruction END qui figure

```
# Jeu de la découverte

GOSUB init                                  # Initialisation des tables et constantes

repeat {                                    # Boucle principale
     GOSUB param                            # Choix et paramétrage du jeu
     repeat {                               # Exécution du jeu
          GOSUB reflexion                   # Choix de la solution secrète

          PRINT : PRINT IN$ : PRINT         # Indication du nom du jeu
          G = 0                             # Initialise le compteur de coups

          repeat {                          # Boucle de traitement du coup

               GOSUB coup : G = G + 1       # Le joueur joue un coup
               GOSUB comparer               # Le programme contrôle le coup

               GOSUB indications            # Le programme donne des indications

               } until (IP = N OR IP = - 1) # Fin de partie
          GOSUB statis                      # Affiche le score
          GOSUB suivant                     # Attente du signal pour démarrer étape suivante

          } until (NX$ = "N" OR NX$ = "E")  # "N" pour passer à un Nouveau jeu

     } until (NX$ = "E")                    # "E" (End) pour finir de jouer

END


100  GOSUB 670
110  GOSUB 590
120  GOSUB 200:PRINT:PRINT IN$:PRINT
130  G=0
140  GOSUB 360:G=G+1:GOSUB 410:GOSUB 480
150  IF IP<> N AND IP<>-1 THEN 140
160  GOSUB 520:GOSUB 710
170  IF NX$="N" THEN 110
180  IF NX$="E" THEN END
190  GOTO 120
```

Voici la « routine » principale du programme du Jeu de la Découverte. La description en BASIC Libre est donnée en premier, suivie des instructions BASIC réelles.

Figure 2.2 : Le Jeu de la Découverte

après THEN en ligne 180 est en dehors de la boucle extérieure et n'est exécutée qu'une seule fois. Si le joueur frappe " E " à la fin d'une partie, le programme sort des deux boucles, médiane et extérieure, et met fin au traitement en exécutant l'instruction END.

Maintenant que nous avons vu quelle est la structure générale du programme, voyons plus en détail les instructions réelles et les sous-programmes. La première ligne est un appel à la routine d'initialisation qui occupe les lignes 670 à 700. Bien que les méthodes de programmation modernes (dont Yourdon et d'autres sont les défenseurs) déconseillent cette façon de faire, de nombreux programmeurs trouvent qu'il est pratique de mettre en tête de programme un appel à une routine qui dimensionne les tables, précise les valeurs des constantes et règle les autres problèmes d'initialisation. Cette routine sera écrite après que le reste du programme soit réalisé, car, lorsqu'on commence à écrire un programme, on sait qu'il y aura beaucoup de choses à faire au stade de l'initialisation, mais on ne peut savoir exactement quelles seront ces tâches, avant que le programme ne soit écrit. Nous examinerons ce sous-programme (présenté à la figure 2.10) ultérieurement.

La ligne suivante contient l'instruction

```
GOSUB 590
```

Cette instruction appelle la subroutine qui précise le type de jeu choisi, et affecte des valeurs à certaines variables. Ces variables conservent la même valeur tant qu'on continue à jouer au même jeu, mais cette valeur change avec le genre du jeu. Nous verrons cette subroutine (présentée en figure 2.9) plus tard ; pour comprendre le programme principal, il nous suffit de savoir que ce sous-programme initialise les variables suivantes :

GM (GAME/Type de Jeu) : est mise à 1 s'il s'agit de trouver un nombre, à 2 s'il s'agit d'un mot.

N (Number of digits/nombre de chiffres) : est rendu égal au nombre de chiffres si le jeu porte sur un nombre. Mais, si l'on joue au Mot, la routine qui saisit le mot « secret » choisi par le second joueur, rendra N égal à la longueur de ce mot.

MG (Maximum Guesses/nombre maximum de coups (suppositions)) initialisé au nombre de coups autorisé. Si l'on joue au Mot, MG est initialisé en même temps que N.

VG (Very Good/Très bien (excellent)) initialisé à une valeur telle que le joueur la considère comme un résultat excellent. Si l'on joue au mot, VG est initialisé juste après N et MG.

La ligne 120 qui suit, comporte quatre instructions :

```
GOSUB 200 : PRINT : PRINT IN$ : PRINT
```

L'instruction GOSUB 200 transfère l'exécution à un sous-programme (fig. 2.3) qui élabore l'exercice suivant. Nous l'étudierons plus en détail ultérieurement. Pour l'instant, nous avons seulement besoin de savoir que cette routine initialise la chaîne de caractères IN$ et code l'exercice de façon compréhensible pour la routine de vérification des réponses du joueur.

Les trois instructions suivantes de la ligne 120 sont :

PRINT : PRINT IN$: PRINT

Ces instructions entraînent l'affichage du texte stocké dans IN$ (par la routine de la ligne 200) précédé et suivi d'une ligne blanche. Ce message, destiné au joueur, l'informe sur la nature et la taille de ce qu'il doit trouver. Par exemple, si Quatre est le jeu choisi, la chaîne de caractères stockée dans IN$ sera

"TROUVER DES NOMBRES DE 4 CHIFFRES"

L'instruction

G = 0

à la ligne 130, initialise le compteur qui enregistre le nombre de coups joués.

A la ligne 140, nous trouvons la structure de la boucle intérieure. Elle contient quatre instructions :

GOSUB 360 : G = G + 1 : GOSUB 410 : GOSUB 480

En début de ligne, l'instruction GOSUB 360 passe la main à la routine qui saisit le coup du joueur. Cette routine (fig. 2.5) accepte l'essai du joueur sous forme de chaîne de caractères (G$) et répartit ensuite ceux-ci dans une table (GC).

Sur cette ligne 140, nous trouvons ensuite l'instruction

G = G + 1

Souvenez-vous qu'avant d'entrer dans la boucle qui commence à la ligne 140, nous avons initialisé G à zéro (ligne 130). Chaque fois que nous appelons le sous-programme de la ligne 360 (pour saisir une solution proposée), nous ajoutons 1 au nombre stocké en G. L'instruction G = G + 1 veut dire « ajouter 1 au nombre stocké en G ». Il en résulte que ce nombre est celui des coups déjà joués.

La troisième instruction de la ligne 140 est un appel à la routine de la ligne 410. Cette routine (fig. 2.6) compare la solution proposée au résultat à trouver, en se servant d'une table codée, dite table des « coups ». Pour poursuivre l'étude du programme principal, tout ce que nous avons besoin de savoir au sujet de ce sous-programme,

est qu'il affecte des valeurs aux variables RG et IP. RG (pour RIGHT, exact) est le nombre de caractères (lettres ou chiffres) présents dans la solution. IP (pour IN PLACE, en place) correspond au nombre de caractères en position correcte.

L'instruction finale de la ligne 140, est aussi l'instruction finale du « corps » de la boucle intérieure. Cette instruction appelle le sous-programme de la ligne 480. Cette routine (fig. 2.7) utilise les valeurs attribuées à IP et RG par la routine de la ligne 410, pour en tirer une « indication » de la forme

EXACTS : 3   EN PLACE : 2

c'est cette même routine qui écrit :

C'EST EXACT

ou

COUPS TROP NOMBREUX — C'EST PERDU

Le premier de ces messages est affiché si IP = N (autrement dit, si tous les chiffres sont à leur place) ; le deuxième correspond à G = MG (ce qui signifie que le nombre maximum d'essais a été atteint). Dans ce dernier cas, IP est forcé à – 1, ce qui ordonne au programme principal de sortir de la boucle qui saisit et contrôle les coups du joueur.

Le test figurant en ligne 150 est la condition d'exécution de la boucle commençant en 140. L'Instruction

IF IP < > N AND IP < > – 1 THEN 140

veut dire « Si le joueur n'a pas encore trouvé (IP < > N) et s'il n'a pas épuisé le nombre de coups autorisé (IP < > – 1) il faut alors revenir en ligne 140 pour exécuter une autre itération de la boucle de traitement du coup ». (Le symbole < > utilisé dans cette instruction signifie « différent de »).

La ligne 160 contient deux instructions :

GOSUB 520 : GOSUB 710

les deux instructions assurent le traitement de la fin de partie. La première appelle le sous-programme de la ligne 520. Cette routine (fig. 2.8) affiche le « score » de la partie. Grâce au compteur de coups G, initialisé à la ligne 130 et mis à jour à chaque itération de la ligne 140, ce sous-programme fait une déclaration du genre

VOUS L'AVEZ TROUVE EN 4 COUPS

Si G est inférieur ou égal à VG, la routine ajoutera

EXCELLENT

Une petite variante (la variété est un élément très important du « comportement » d'un programme de jeu) sera

VOUS DEVEZ ETRE EXTRA-LUCIDE

dans le cas où le joueur trouve à son premier essai. Ce message permet par la même occasion, d'éviter les instructions nécessaires pour éviter la faute d'orthographe suivante :

VOUS L'AVEZ TROUVE EN 1 COUPS

La seconde instruction de la ligne 160 appelle le sous-programme de la ligne 710. Ce sous-programme (fig. 2.10) saisit un caractère isolé, frappé par le joueur et le stocke dans NX$. C'est ainsi que le joueur communique ses directives au programme et lui indique ce qu'il convient de faire. Les choix sont interprétés par les lignes 170, 180 et 190 de la façon suivante :

N	Donnez au joueur la possibilité de changer de Jeu.
E	Fin de l'utilisation du programme de jeu et retour sous BASIC.
autres touches :	Recommencez une nouvelle partie du même jeu.

L'utilisation d'une entrée consistant en un caractère isolé permet de poursuivre de façon souple. Par exemple, supposons que vous veniez de faire une partie de Quatre. Le programme vient d'annoncer :

C'EST EXACT ! VOUS L'AVEZ TROUVE EN 8 COUPS

Le reste de l'écran n'a pas été modifié et vous réexaminez les coups que vous avez joués, pour voir comment vous auriez pu gagner en faisant moins d'essais. Quand vous avez fini d'analyser la partie, vous appuyez sur la barre d'espacement. L'écran est effacé et une nouvelle partie de Quatre commence. Le programme vous laisse donc étudier la partie précédente pendant tout le temps que vous désirez ; mais, d'autre part, vous pourrez commencer une autre partie sans avoir à répondre à une question du genre :

DESIREZ-VOUS FAIRE UNE AUTRE PARTIE (O OU N) ?

Quand vous appuyez sur la barre d'espacement, comme nous venons de le décrire, c'est vous qui avez l'initiative du déroulement du programme. Quand il vous faut répondre à une question comme la

```
# Création du problème

reflexion   GOSUB effacecran                          # écran effacé pour début jeu
            FOR ZZ = BH TO TH                         # r.a.z. tableau « coups »
                 HT(ZZ) = 0
                 NEXT ZZ
            ON GM GOSUB nombre, demandermot
            RETURN

# Imaginer un nombre de N chiffres, tous différents

nombre      FOR ZZ = 1 TO N

                 repeat
                       HH = INT(RND(1) * 10) + L      # ASCII du chiffre
                       until (HT(HH) = 0)             # doublons interdits
                 HT(HH) = HT(HH) + 2↑(ZZ - 1)         # définir la puissance de 2
                                                      pour cette position
                 NEXT ZZ
            IN$ = "TROUVER DES NOMBRES DE" + STR$(N) + "CHIFFRES."
            GL = N                                    # nombre prévu
            RETURN
```

```
200    GOSUB 740
210    FOR ZZ=BH TO TH:HT(ZZ)=0:NEXT ZZ
220    ON GM GOSUB 310,240
230    RETURN
310    FOR ZZ=1 TO N
320*   HH=INT(RND(1)*10)+L:IF HT(HH)<>0 THEN 320
330    HT(HH)=HT(HH)+2↑(ZZ-1):NEXT ZZ
340**  INS="TROUVER DES NOMBRES DE"+STR$(N)+"CHIFFRES":GL=N
350    RETURN
```

Voici le sous-programme qui crée un nouveau problème. On voit également l'une des deux routines appelées, celle qui fabrique un nombre de N chiffres dont tous les chiffres sont différents. Chaque routine est présentée d'abord en BASIC Libre, ensuite en BASIC.

---

* Sur TRS-80, le RND(0) doit être remplacé par RND(1).

** La version Apple de cette ligne ne diffère que par le nombre d'espaces dans les constantes.

Figure 2.3 : Jeu de la Découverte — Routine « réflexion »

précédente, c'est le programme qui a l'initiative. Faire en sorte que le joueur garde le contrôle du déroulement du jeu, contribue à l'agrément de ce dernier.

Et maintenant, supposons que vous désiriez passer du jeu de Quatre à celui de Cinq. Au lieu d'appuyer sur la barre d'espacement en fin de partie, vous tapez N. L'écran est affacé et le programme vous offre le choix entre plusieurs types de jeu. Comparez cela avec le dialogue ci-après :

DESIREZ-VOUS FAIRE UNE AUTRE PARTIE (O OU N) ? **N**

DESIREZ-VOUS JOUER A UN AUTRE JEU (O OU N) ?

Bien sûr, tout ceci est une question de convenance personnelle. Si vous préférez le dialogue ci-dessus, vous pourriez trouver intéressant l'exercice consistant à modifier le programme en conséquence. Les seules modifications nécessaires consistent à supprimer l'instruction GOSUB 710 figurant à la ligne 160 et à remplacer les lignes 170, 180 et 190, par les instructions adéquates. Vous n'avez besoin de connaître le fonctionnement d'aucune routine pour effectuer cette modification.

Maintenant que nous sommes arrivés à la fin du programme du Jeu de la Découverte de la figure 2.2, regardons les sous-programmes plus en détail. Nous commencerons avec celui de la ligne 200 (fig. 2.3) et les deux routines situées en lignes 310 et 240 qui lui sont subordonnées (voir figures 2.3 et 2.4).

Le sous-programme de la ligne 200 élabore les données d'une nouvelle partie. Par exemple, si l'on joue à Quatre, c'est cette routine qui tire le numéro à quatre chiffres que le joueur va s'efforcer de trouver. Si l'on joue au Mot, c'est elle également qui demande au second joueur de fournir un mot que le premier devra trouvé. Après avoir stocké en mémoire un nombre ou un mot, la routine, le code, en utilisant un tableau des « coups » appelé HT (Hit, coup), dont nous allons parler bientôt. Le but de ce codage est de faciliter le contrôle de la validité des coups du joueur.

Un moyen évident d'évaluer cette validité serait la suivante. Chaque caractère du coup joué serait d'abord comparé au caractère situé à la même place dans la solution. En cas d'égalité, les variables IP et RG seraient incrémentées, et le programme passerait alors au caractère suivant. Dans le cas contraire, le programme balaierait alors les autres positions de la solution pour voir si le caractère s'y trouve. Dans l'affirmative, RG serait incrémenté et le programme passerait à l'examen du caractère suivant du coup joué.

Peut-être cela vous intéressera-t-il de remplacer le sous-programme de la figure 2.6 par une routine utilisant l'algorithme ci-

dessus. Il est probable que vous vous aperceviez que votre sous-programme met un temps appréciable pour effectuer ce contrôle. Si vous décidiez de mener à bien cet exercice, il vous faudrait modifier la routine de la ligne 310 de façon à mettre sous forme de chaîne de caractères, le nombre de quatre chiffres qui y est créé. Cette chaîne devrait alors être sauvegardée dans la variable « chaîne de carac-

```
# Demander un mot pour une nouvelle partie

demandermot   repeat {                                    # entrer chaîne
                  INPUT " MOT SECRET "; H$
                  N = LEN(H$)
                  } until (0 < N AND N < = MX)
              IN$ = " DECOUVERTE MOTS DE " + STR$(N) + " LETTRES. "
              GL = N                                      # longueur prévue coup
              FOR ZZ = 1 TO N                             # codage car. dans table
                                                          « coups »
                  HH = ASC(MID$(H$, ZZ, 1))
                  HT(HH) = HT(HH) + 2↑(ZZ − 1)
                  NEXT ZZ
              VG = N                                      # « excellent » pour peu de
                                                          coups
              MG = ncoupmax                               # « perdu » si trop de coups
              GOSUB effacecran                            # effacer le mot
              RETURN
```

```
240* INPUT "MOT SECRET";H$:N=LEN(H$)
250  IF N=0 OR N>MX THEN 240
260* IN$="DECOUVERTE MOTS DE"+STR$(N)+"LETTRES":GL=N
270  FOR ZZ=1 TO N:HH=ASC(MID$(H$,ZZ,1))
280  HT(HH)=HT(HH)+2↑(ZZ−1):NEXT ZZ
290  VG=N:MG=20
300  GOSUB 740:RETURN
```

Voici l'autre routine appelée par la première des routines de la figure 2.3. Elle demande à l'un des joueurs de fournir le mot que l'autre devra découvrir. Cette routine est présentée d'abord en BASIC Libre, puis en BASIC.

---

* Les versions Apple de ces lignes ne diffèrent que par la valeur des constantes littérales.

Figure 2.4 : Jeu de la Découverte — Routine « demandermot »

tères » H$. Le sous-programme de la ligne 240 utilise déjà H$ pour stocker le mot indiqué par le second joueur.

La méthode de codage qui est en réalité utilisée pour résoudre ce problème, assure un contrôle rapide des coups joués. On construit une table baptisée HT. La taille de HT est égale au nombre des caractères ASCII possibles : 128 pour l'Apple et le TRS-80, 256 pour le Pet. (Le Pet utilise un jeu étendu de caractères ASCII comprenant des caractères graphiques qu'on peut entrer à partir du clavier.)

Au départ, la table HT est mise à zéro. Puis, pour chacun des caractères du nombre ou mot à trouver, on va marquer le tableau HT à la position correspondant au code ASCII concerné. Par exemple, supposons que le jeu en cours soit Quatre et que le nombre créé par le programme soit 4726. Le tableau HT sera marqué aux positions 52, 55, 50 et 54, car 52 est le code ASCII de « 4 », 55 celui de « 7 », 50 celui de « 2 » et enfin 54 celui de « 6 ». (La liste des codes ASCII est donnée en Appendice A.)

Marquer la table HT aux positions correspondant aux codes ASCII des caractères de la solution, ne fournira aucun renseignement sur la place occupée par les différents caractères. Par exemple, les deux nombres 4726 et 2674 donneraient lieu au marquage des quatre mêmes positions du tableau HT. Etant donné que nous avons besoin d'être renseignés sur la position occupée par chacun des chiffres, les marquages que nous faisons dans le tableau des coups doivent préciser les positions des caractères composant la solution. Par exemple, pour 4276, nous pourrions mettre un 1 dans HT(52), un 2 dans HT(55), un 3 dans HT(50) et un 4 dans HT(54). Pour 2674, nous mettrions un 1 en HT(50), un 2 en HT(54), un 3 en HT(55) et un 4 en HT(52). Ainsi, chaque élément du tableau HT contiendrait un nombre donnant la position occupée dans la solution, par le caractère correspondant.

Cette méthode conviendrait parfaitement pour Quatre et les autres jeux de nombres, car dans ces jeux le même chiffre ne peut pas figurer plusieurs fois. Mais pour le jeu du Mot, elle ne serait pas valable puisqu'une même lettre peut se trouver en plusieurs positions différentes. Par exemple, si le mot retenu est BILL, l'élément HT(76) (qui correspond à la lettre L) devrait contenir à la fois 3 et 4. En effet, la lettre L figure simultanément à la troisième et quatrième position du mot BILL. Or, nous pouvons stocker à la fois 3 et 4 en HT(76) d'une façon simple en affectant à HT(76) la valeur 24, c'est-à-dire $2^3 + 2^4$.

Vous savez probablement que tout nombre entier (par exemple 24) peut être décomposé et ceci d'une seule et unique façon : en une

somme de puissances de deux. C'est pourquoi, si HT(76) a la valeur 24, nous pouvons par le calcul trouver que $24 = 2^3 + 2^4$, ce qui nous informe que 3 et 4 sont les positions occupées par la lettre L dans le mot BILL. Les instructions qui font ces calculs font partie du sous-programme de la ligne 410 (voir figure 2.6).

Le sous-programme de la ligne 200 commence par mettre à zéro, les éléments du tableau HT qui sont concernés par le jeu. Par exemple, pour Quatre et les autres jeux sur les nombres, les dix éléments HT(48) à HT(57) (correspondant aux codes ASCII des caractères 0, 1, 2, ..., 9) seront mis à zéro. Les variables BH (« bottom hit », coup le plus bas) et TH (« Top hit », coup le plus haut), qui précisent les bornes de cette suite d'indices, sont initialisées par la routine de la figure 2.9. Pour Quatre par exemple, on aura BH = 48 et TH = 57. Nous ne mettrons à zéro que les éléments de HT qui sont nécessaires au jeu, parce que la mise à zéro de la totalité des 256 éléments de la table peut demander un temps appréciable.

Après avoir « effacé » la partie concernée du tableau HT, la routine exécute l'instruction :

```
ON GM GOSUB 310,240
```

Cette instruction déclenche un appel, soit au sous-programme de la ligne 310, soit à celui de la ligne 240, et ceci, en fonction de la valeur de la variable GM. La valeur de GM est fixée par la subroutine de la figure 2.9. Cette valeur est 1 pour tous les jeux de nombres (par exemple, Quatre) et 2 pour le jeu du Mot. Il en résulte que la routine de la ligne 310 (fig. 2.3) est appelée pour les nombres et celle de la ligne 240 (fig. 2.4) pour le Mot. Ces deux sous-programmes sont courts et faciles à comprendre, mais plusieurs points méritent d'être développés.

Tout d'abord, portons notre attention sur la ligne 320 de la routine utilisée pour les jeux de nombres :

```
320   HH = INT(RND(1)*10) + L : IF HT(HH) <> O THEN 320
```

La première instruction de la ligne 320 tire au hasard un nombre entier parmi les dix valeurs, commençant par celle qui est stockée dans la variable L. Comme la routine d'initialisation a mis L à 48 qui est le code ASCII de zéro, le tirage s'effectuera dans la fourchette des dix nombres commençant à 48. En raison de la disposition du jeu de caractères ASCII, ces dix nombres sont les codes ASCII des chiffres 0, 1, ..., 9.

La deuxième instruction de la ligne 320 regarde quelle est la valeur de l'élément de HT correspondant au code ASCII concerné. Si cette

valeur est différente de zéro, le programme revient boucler en 320 pour recommencer le tirage. Cette façon de faire donne la certitude que le nombre ainsi créé, n'aura pas deux chiffres identiques.

La ligne suivante, numérotée 330, contient l'instruction qui va stocker la puissance de deux, adéquate dans l'élément de la table HT qui correspond au code ASCII qui vient d'être tiré :

HT(HH) = HT(HH) + 2↑(ZZ − 1)

Etant donné qu'à la ligne précédente on vient de s'assurer que HT(HH) était bien égal à zéro, cette dernière instruction est équivalente à

HT(HH) = 2↑(ZZ − 1)

La valeur de ZZ, indice de la boucle FOR... NEXT, progresse de 1 à N, N étant le nombre des chiffres (par exemple 4 chiffres pour Quatre). Ainsi, si 50 (code ASCII de 2) est le premier code à être tiré à la ligne 320, nous aurons ZZ = 1 et HH = 50. Par conséquent, l'instruction précédente revient à écrire

HT(50) = 2↑(0)

Si 54 (code de 6) est le deuxième code tiré, l'instruction devient alors

HT(54) = 2↑(1)

Si l'on joue à Quatre, avec comme solution à trouver le nombre 2649, HT(50) contiendra 1 (car 2↑0 = 1), HT(54) contiendra 2, HT(52) contiendra 4 et enfin HT(57) contiendra 8. Les six autres éléments (sur les dix allant de HT(48) à HT(57) qui ont été mis à zéro par la routine de la ligne 200, restent à zéro.

Les dernières fonctions assurées par la routine de la ligne 310 sont de préciser d'une part, la valeur de IN$ (il s'agit d'un message de « rappel » qui est affiché en début de partie chaque fois que le joueur propose une réponse de longueur incorrecte), et d'autre part la valeur de GL (Guess Length, nombre de caractères prévu pour les réponses du joueur).

Le sous-programme de la ligne 240 (fig. 2.4) ressemble à la routine de la ligne 310. Notons que les lignes 270 et 280 comportent les instructions

HH = ASC(MID$(H$, ZZ, 1))
HT(HH) = HT(HH) + 2↑(ZZ − 1)

La première montre comment le code ASCII de chaque caractère est extrait du mot à découvrir, caractère par caractère. L'expression

MID$(H$, ZZ, 1)

représente une sous-chaîne d'un seul caractère (à cause de la valeur 1 du troisième argument) prise dans H$ (qui est le premier argument) et commençant en position ZZ. Or, ZZ, étant la variable de contrôle de la boucle FOR...NEXT, prend les valeurs de 1 à N, N étant le nombre de caractères de H$ (du fait de l'instruction N = LEN(H$) à la ligne 240). Ainsi MIDS(H$, ZZ, 1) représente le ZZ-ième caractère de H$ sous forme d'un caractère unique. La fonction ASC transforme ce caractère en nombre, ce nombre étant le code ASCII correspondant. Il est mis dans HH, qui devient ainsi l'indice qui pointe vers le tableau HT.

L'instruction

```
HT(HH) = HH(HH) + 2↑(ZZ - 1)
```

de la ligne 280 est la même que celle qui se trouve en ligne 330 (voir figure 2.3). Lorsque nous avons étudié la ligne 330, nous avons noté que, sachant que HT(HH) était égal à zéro, il en découlait que l'instruction était équivalente à

```
HT(HH) = 2↑(ZZ - 1)
```

Dans le cas de la ligne 280, cela n'est plus vrai, car le même caractère (et par conséquent, une valeur de HH identique) a pu apparaître précédemment dans le mot secret. Aussi, la puissance de deux (2↑(ZZ – 1)) est ajoutée au contenu précédent de HT(HH). Ce contenu précédent est zéro si le caractère considéré n'est pas encore apparu dans le mot. Dans le cas contraire HT(HH) contiendra une somme de termes de la forme 2↑(ZZ – 1) où les valeurs de ZZ correspondent aux positions pour lesquelles le caractère est déjà apparu.

Le reste de la routine de la figure 2.4 n'a pas besoin d'être expliquée. Notons simplement qu'elle se termine par un appel à la subroutine de la ligne 740 (fig. 2.12) pour effacer l'écran. L'effacement de l'écran permet de faire disparaître le mot secret. Dans la plupart des cas, il permet aussi d'afficher le dialogue du jeu sans effet de rouleau (scrolling), c'est-à-dire sans disparition de la ligne supérieure chaque fois qu'une nouvelle ligne est affichée dans le bas de l'écran.

La figure 2.5 présente le sous-programme qui demande au joueur sa réponse et la saisit. Les deux premières lignes (360 et 370) contiennent les instructions

```
360  INPUT "QUE PROPOSEZ-VOUS"; G$
     IF LEN(G$) <> GL THEN PRINT IN$: GOTO 360
```

Ces instructions seront répétées jusqu'à ce que le joueur ait frappé une réponse d'une longueur qui soit exactement celle spécifiée dans GL (par exemple, quatre pour Quatre). La chaîne IN$ (c'est-à-dire

le message de « rappel » du début) sera réaffichée chaque fois que cette longueur n'est pas respectée. Le joueur est alors invité à reformuler sa réponse. Il faut remarquer que les réponses refusées ne sont pas ajoutées au total des coups joués. En effet, la variable G n'est incrémentée qu'au retour de cette routine (voir la ligne 140 à la figure 2.2).

Les deux lignes suivantes, 380 et 390, contiennent les instructions qui décomposent la réponse du joueur en caractères séparés dont les codes ASCII sont alors stockés dans le tableau GC (comme Guess Characters, caractères de la supposition) :

```
FOR ZZ = 1 TO GL
GC(ZZ) = ASC(MID$(G$,ZZ,1)) : NEXT ZZ
```

Ces instructions sont semblables à celles qui ont été utilisées pour analyser le mot à découvrir (voir ligne 270, figure 2.4).

```
# Entrée du coup joué

coup   repeat {
           INPUT "QUE PROPOSEZ-VOUS"; G$
           IF LEN(G$) <> GL THEN
                PRINT IN$
           } until (LEN(G$) = GL)
       FOR ZZ = 1 TO GL                      # carac. coup dans la table GC
           GC(ZZ) = ASC(MID$(G$, ZZ, 1))
           NEXT ZZ
       RETURN
```

```
360* INPUT "QUE PROPOSEZ-VOUS";G$
370  IF LEN(G$)<>GL THEN PRINT IN$:GOTO 360
380  FOR ZZ=1 TO GL
390  GC(ZZ)=ASC(MID$(G$,ZZ,1)):NEXT ZZ
400  RETURN
```

Cette routine demande que le joueur joue un coup et saisit sa proposition. Les caractères du coup sont alors mis un par un dans un tableau. La routine est présentée d'abord en BASIC Libre, puis en BASIC.

* La version Apple de cette ligne comporte un point d'interrogation dans la constante littérale.

Figure 2.5 : Jeu de la Découverte — Routine « coup »

L'essai du joueur (maintenant encodé dans la table GC) est comparé à la solution (encodée dans HT) par le sous-programme de la ligne 410. Ce sous-programme est présenté à la figure 2.6. La routine commence par mettre à zéro les compteurs RG (nombre de caractères corrects) et IP (nombre de caractères bien placés) (ligne 410). Le reste de la routine est constituée par une boucle dont le corps va de la ligne 420 à la ligne 450. C'est une boucle FOR ... NEXT contrôlée par la variable ZZ. Dans la boucle, chaque

```
# Comparer le coup au mot ou au nombre secret

comparer  RF = 0                          # départ avec aucun exact
                                          aucun en place
          IP = 0
          FOR ZZ = 1 TO GL                # comparer le coup au tableau
                                          des « coups »
              HH = HT(GC(ZZ))
              IF HH < > 0 THEN {          # le caractère est dans la chaîne
                  RG = RG + 1
                  HH = INT(HH/2↑(ZZ − 1)) # cherchons la puissance de
                                          2 correspondante
                  IF HH est impair THEN   # caractère bien placé
                      IP = IP + 1
                  }
              NEXT ZZ
          RETURN
```

```
410 RG=0:IP=0
420 FOR ZZ=1 TO GL:HH=HT(GC(ZZ))
430 IF HH=0 THEN 460
440 RG=RG+1:HH=INT(HH/2↑(ZZ−1))
450 IF HH< >2*INT(HH/2) THEN IP=IP+1
460 NEXT ZZ
470 RETURN
```

Cette routine compare le tableau GC qui contient les caractères de coup (tableau valorisé à partir du coup joué par la routine de la figure 2.5) au tableau HT dit des « coups ». HT a autant d'éléments qu'il y a de caractères ASCII possibles. Si le caractère ne fait pas partie de la solution, la valeur de l'élément est zéro.

Dans le cas contraire la valeur de l'élément de HT est la somme des puissances de 2 correspondant aux différentes positions occupées par le caractère.

La routine est présentée d'abord en BASIC Libre, puis en BASIC.

Figure 2.6 : Jeu de la Découverte — Routine « comparer »

caractère de la réponse (caractères stockés dans les éléments 1 à GL de la table GC) est rapproché du contenu de l'élément correspondant du tableau des coups HT. L'instruction

```
HH = HT(GC(ZZ))
```

à la ligne 420, fournit la valeur de l'élément de HT qui correspond au ZZ-ième caractère de la réponse. Si HH est nul, c'est que le caractère ne figure pas dans la solution du problème. Si HH n'est pas nul, c'est qu'au contraire le caractère y figure. Dans ce dernier cas, RG est incrémenté de 1 (ligne 440). Le programme regarde alors si la valeur 2↑(ZZ − 1) est ou n'est pas l'une des puissances de 2 faisant partie de la valeur de HH. Si oui, IP est incrémenté. Les instructions qui se chargent de ce travail, sont aux lignes 440 et 450 :

```
HH = INT(HH/2↑(ZZ − 1))
IF HH <> 2 * INT(HH/2) THEN IP = IP + 1
```

La première « laisse tomber » toutes les puissances de 2 (s'il y en a) inférieures à 2↑(ZZ − 1). Si vous connaissez les instructions disponibles en langage machine, vous pouvez assimiler celle-ci à un décalage à droite qui amènerait le bit qui nous intéresse à l'extrême droite (position du bit le moins significatif). Si vous préférez penser en termes mathématiques, la valeur initiale de HH est de la forme

$$a_0 + a_1 \times 2 + \cdots + a_{GL-1} \times 2^{GL-1}$$

où $a_0, a_1, \ldots, a_{GL-1}$ prennent les valeurs 0 ou 1. Les indices 0 à GL − 1 correspondent aux valeurs 1 à GL de la variable ZZ. En conséquence, l'opération pratiquée sur HH remplace la valeur d'origine par la valeur :

$$a_{ZZ-1} + (a_{ZZ} \times 2 + \cdots + a_{GL-1} \times 2^{GL-ZZ})$$

quand ZZ < GL et par a_{GL-1} quand ZZ = GL. La valeur du terme entre parenthèses est toujours paire, de sorte que la valeur de a_{ZZ-1} peut être trouvée de façon simple en déterminant si la nouvelle valeur de HH est paire (auquel cas $a_{ZZ-1} = 0$) ou impaire (auquel cas $a_{ZZ-1} = 1$). La condition

```
HH<>2 * INT(HH/2)
```

revient à dire « la valeur de HH est impaire ».

Même si vous avez eu quelques difficultés à suivre ces commentaires, ne vous inquiétez pas, il n'est pas nécessaire de connaître les détails de cette opération pour comprendre ce qui va suivre.

La routine de la figure 2.7 a déjà été commentée de façon succincte. Le point principal dont il convient de se souvenir, est qu'elle force IP à − 1 si le joueur a épuisé son crédit de coups.

La routine de la figure 2.8 est également très simple, mais elle réalise une fonction importante qui peut facilement être perfectionnée et étendue. En effet, le sous-programme porte une appréciation sur votre façon de jouer et vous donne une tape amicale dans le dos si vous avez été particulièrement brillant. Elle pourrait (mais ne le fait pas) comparer vos résultats présents à ceux de parties précédentes. Une routine qui utilise un tel effet de feed-back rend le jeu beaucoup plus amusant.

```
# Donner une indication (ou annoncer le gain ou la perte de la partie)

indication  IF IP = N THEN                              # tous en place
                  PRINT " C'EST EXACT! ";
            else IF G = MG THEN {                       # maximum atteint
                  PRINT "COUPS TROP NOMBREUX— C'EST PERDU! "
                  IP = - 1                              # signal « partie terminée »
                  }
            else
                  PRINT " EXACTS : "; RG, " EN PLACE : "; IP
            RETURN
```

```
480  IF IP=N THEN PRINT "C'EST EXACT." ;:GOTO 510
490  IF G=MG THEN PRINT "COUPS TROP NOMBREUX—C'EST PERDU!":IP=-1:GOTO
     510
500* PRINT "EXACTS:"; RG, "EN PLACE!"; IP
510  RETURN
```

Cette routine regarde si le joueur a trouvé la solution ou a épuisé son crédit de coups. Si aucune de ces deux conditions n'est vérifiée, la routine affiche le nombre de caractères exacts et le nombre des caractères en place. Cette même routine force IP à − 1 si le joueur a épuisé son crédit de coups. La routine est présentée en BASIC Libre ; puis en BASIC.

* La version Apple de cette ligne en diffère par le nombre d'espaces dans les constantes et l'utilisation de TAB(11) à la place de la virgule placée après RG.

Figure 2.7 : Jeu de la Découverte — Routine « indication »

Le sous-programme de la figure 2.9 sert à choisir le jeu désiré. Cette routine est appelée en début de programme et chaque fois que le joueur appuie sur « N » à la fin d'une partie. Après avoir effacé l'écran par un appel à la routine de la ligne 740, le sous-programme pose la question "JEU :" et saisit le caractère unique que vous frappez en réponse. Si ce caractère est un chiffre de 1 à 9, c'est que vous choisissez de jouer au jeu de nombre correspondant. Un « W » choisira le jeu du Mot. Tout autre caractère aboutit à la sélection du jeu Quatre. Quatre est donc le jeu choisi par *défaut*.

```
# Statistiques sur partie juste terminée

statis  IF IP = N THEN {                          # tous en place
          IF G = 1 THEN                           # trouvé du premier coup
            PRINT "VOUS DEVEZ ETRE EXTRA-LUCIDE."
          else {
              PRINT "VOUS L'AVEZ TROUVE EN"; G; "COUPS."
              IF G < = VG THEN                    # trouvé en très peu de coups
                  {PRINT : PRINT "EXCELLENT."}
              }
          }
        else IF IP = - 1 THEN                     # crédit de coup épuisé
          PRINT "VOUS NE L'AVEZ PAS TROUVE EN"; MG; "ESSAIS."
        RETURN
```

```
520  IF IP<>N THEN 570
530  IF G=1 THEN PRINT"VOUS DEVEZ ETRE EXTRA-LUCIDE.":GOTO 580
540* PRINT"VOUS L'AVEZ TROUVE EN";G;"COUPS."
550  IF G<=VG THEN PRINT:PRINT "EXCELLENT."
560  GOTO 580
570* IF IP=-1 THEN PRINT "VOUS NE L'AVEZ PAS TROUVE EN";MG;"ESSAIS."
580  RETURN
```

Cette routine est appelée en fin de partie. Ou bien le joueur a trouvé la solution, ou bien la partie est terminée parce que le joueur a épuisé son crédit de coups. Dans ce dernier cas, il y a d'ailleurs une omission flagrante dans le programme.

La routine est présentée d'abord en BASIC Libre, puis en BASIC.

---

* La version Apple de ces lignes ne diffère que par le nombre d'espaces dans les constantes littérales.

Figure 2.8 : Jeu de la Découverte — Routine « statis »

\# Paramétrage nouvelle partie

```
param       GOSUB effacecran
            PRINT "JEU:";
            GOSUB uncar : N = VAL(X$)          # code du jeu
            IF N <> 0 THEN                     # un chiffre signifie un jeu
                                               de nombre
                 GM = 1
            else IF X$ = "W" THEN              # si "W" jeu de Mot
                 GM = 2
            else
                 { N = 4 : GM = 1 }            # par défaut jeu de Quatre
            ON GM GOSUB paranombre, paramot
            RETURN

paranombre  VG = N + 2 : MG = 2 * VG           # fixer score « excellent » et
                                               limite
            BH = L : TH = L + 9                # plage à utiliser dans table
                                               « coups »
            RETURN

paramot     VG = 0 : MG = 99                   # VG et MG seront précisés
                                               par demandermot
            BH = O : TH = nombrascii           # plage à utiliser dans table
                                               « coups »
            RETURN
```

```
590  GOSUB 740:PRINT "JEU";:GOSUB 720
600  N=VAL(X$):IF N<>0 THEN GM = 1:GOTO 630
610  IF X$="W"THEN GM=2:GOTO 630
620  N=4:GM=1
630  ON GM GOSUB 650,660
640  RETURN
650  VG=N+2:MG=2*VG:BH=L:TH=L+9:RETURN
660* VG=0:MG=99:BH=0:TH=255:RETURN
```

Voici la routine qui permet au joueur de choisir entre les différents jeux possibles. Trouver un nombre ayant N chiffres ou trouver un mot entré par un autre joueur. La plus grande partie de la routine a pour seul but de suggérer des possibilités (par exemple, on pourrait trouver mieux pour fixer les valeurs de VG et de MG), et pour fournir un cadre facilitant l'addition d'autres types de jeu.

La routine est présentée en BASIC Libre, puis en BASIC.

---

* Sur Apple et TRS-80 il convient de remplacer 255 par 127.

Figure 2.9 : Jeu de la Découverte — Routine « param »

Le fonctionnement de cette routine repose sur le fait que la fonction VAL fournit la valeur de toute chaîne de caractères pouvant être interprétée comme un nombre, tandis qu'elle renvoie zéro pour les chaînes dont les caractères ne sont pas numériques (par exemple " W").

Les deux subroutines des lignes 650 et 660 fixent les valeurs de MG (nombre de coups maximum), VG (excellent) ainsi que les limites BH et TH de la table HT. On pourrait imaginer des modes de calcul plus sophistiqués pour MG et VG. Des appréciations supplémentaires s'ajoutant à VG, pourraient être incorporées aux instructions de la figure 2.8 de façon à ce que le joueur puisse après coup recevoir une notation plus sélective.

La variable GM et l'instruction

```
ON GM GOSUB 650,660
```

donnent le cadre dans lequel on peut ajouter facilement les routines de paramétrage d'éventuels jeux supplémentaires. En d'autres termes, cette partie du programme a été conçue pour être facilement généralisée.

Les sous-programmes de la figure 2.10 font partie de la catégorie des « divers », habituellement reléguée en fin de listing.

La première routine (ligne 710) comporte les instructions

```
GOSUB 720 : NX$ = X$ : RETURN
```

On pourrait remplacer cette routine par un appel direct à la routine de la ligne 720 et utiliser X$ directement. Cela impliquerait la modification des lignes 160, 170 et 180 (voir figure 2.2).

Cette routine de la ligne 710 offre un exemple d'un cas fréquemment rencontré dans les programmes bien conçus. Une routine initialement prévue pour remplir une fonction précise et importante, peut finalement se transformer en un simple appel à une autre routine. Au nom de la lisibilité et de l'efficacité, on devrait éliminer de telles routines « intermédiaires », mais comme leur présence n'est pas dangereuse et ne diminue que faiblement la vitesse d'exécution, elles échappent souvent à la mise à mort.

La seconde routine de la figure 2.10 est la routine d'initialisation. Elle devrait normalement se trouver tout à fait en fin de programme, mais cet emplacement a été réservé aux quelques routines du Jeu de la Découverte dont les Versions Apple, Pet et TRS-80 diffèrent de façon significative. La routine d'initialisation dimensionne deux tableaux et affecte une valeur à MX. MX est utilisée dans le sous-programme de la figure 2.4 pour s'assurer que le mot à trouver proposé comme Mot, n'est pas plus long que la table GC. (La

table GC, comme nous l'avons vu, contient les caractères du coup en cours.) Cette routine d'initialisation précise également la valeur L du code ASCII du caractère zéro. Dans une version antérieure du programme, il était logique de préciser L à cet endroit. Maintenant il vaudrait mieux le faire au niveau de la routine de la ligne 650 (voir figure 2.9)

```
# Obtenir code partie suivante

suivant  GOSUB uncar                     # entrer caractère unique
         NX$ = X$                        # le mettre dans NX$
         RETURN

# initialiser

init     DIM HT(nombrascii)              # tableau « coups » indicé par code ASCII
         MX = motmax : DIM GC(MX)        # tableau des caractères du coup
         L = ASC("0")                    # limite inférieure pour choix des chiffres
         RETURN
```

```
710   GOSUB 720:NX$=X$:RETURN
670*  DIM (HT(255)
680   MX=20:DIM GC(MX)
690   L=ASC("0")
700   RETURN
```

Voivi deux petites routines. La première, qui est appelée en fin de chaque partie, renvoie un caractère unique dans la variable NX$. Le programme en déduira l'une des trois choses ci-après : le joueur désire changer de jeu (NX$ = "N"), le joueur ne veut plus jouer (NX$ = "E"), ou bien le joueur désire continuer avec le même jeu (n'importe quel autre caractère). Le principal objet de cette routine est de provoquer, à la fin du jeu, une pause pendant laquelle le joueur pourra revenir sur les différents coups de la partie pour voir ce qu'il peut en apprendre.

La seconde routine est la routine d'initialisation. Les programmeurs un peu « rétro » commencent toujours leurs programmes par un appel à une routine d'initialisation. Celle-ci dimensionne deux tableaux et fixe la valeur de deux constantes.

Les routines sont présentées d'abord en BASIC Libre, puis en BASIC.

* Sur Apple et TRS-80 dimensionner le tableau à 127.

Figure 2.10 : ***Jeu de la Découverte — Deux petites routines***

Entrée d'un caractère unique

Version Pet

```
uncar   repeat {
             X$ = CHR$ (RND(1))
             GET X$
             }until (X$ <> "")
        RETURN

    720   X$=CHR$(RND(1)):GET X$:IF X$="" THEN 720
    730   RETURN
```

Version Apple

```
uncar   GET X$
        IF X$ = CHR$(3) THEN                    # faire marcher control-C
             STOP
        RETURN

    720   GET X$:IF X$=CHR$(3) THEN STOP
    730   RETURN
```

Version TRS-80

```
uncar   repeat {
             X$ = CHR$(RND(0))
             GET X$
             } until (X$<> "")
        RETURN

    720   X$=CHR$(RND(0)):X$=INKEY$:IF X$="" THEN 720
    730   RETURN
```

Voici l'importante opération de saisie d'un caractère unique sans affichage. Le Pet, l'Apple et le TRS-80 la réalisent de trois manières différentes.

Les routines sont présentées en BASIC Libre, puis en BASIC. Notons que les trois versions BASIC ont les mêmes numéros de lignes.

Figure 2.11 : Entrée d'un caractère unique

Les figures 2.11 et 2.12 montrent les routines qui sont différentes pour l'Apple, le Pet et le TRS-80. Toutes les routines précédentes étaient identiques pour ces trois systèmes (mises à part les différences mineures — c'est-à-dire les espaces de part et d'autre d'un nombre, le « ? » de l'instruction INPUT — et le dimensionnement différent du tableau HT).

La première routine qui varie suivant les systèmes, est celle qui assure la saisie d'un caractère isolé (fig. 2.11). Elle met en œuvre des moyens différents selon les systèmes. Par exemple, l'instruction GET du Pet n'attend pas qu'un caractère soit frappé. Cette caractéristique est mise à profit pour placer GET à l'intérieur d'une boucle qui contient aussi un appel à la fonction RND. Comme nous l'avons expliqué au Chapitre 1, cette façon de faire assure un tirage au hasard vraiment aléatoire. La version Apple du Get attend par contre la frappe, de telle sorte qu'il est impossible (ou qu'il n'est pas nécessaire) de l'utiliser dans une boucle. Comme par ailleurs GET ne reconnaît pas le code control-C habituellement utilisé pour interrompre l'exécution du programme, cette possibilité est rajoutée par un test explicite. En effet, CHR$(3) représente un caractère équivalent à control-C. Le TRS-80 utilise la fonction INKEY$ exactement comme le Pet utilise GET.

La routine d'effacement de l'écran (fig. 2.12) ne comporte qu'une seule instruction quelle que soit la version. Sur le Pet, l'écran est effacé en effectuant un « PRINT » du caractère correspondant à la touche CLR. Sur Apple et TRS-80, une instruction spécifique est utilisée. HOME pour Apple, CLS pour TRS-80.

Ceci termine l'analyse du programme du Jeu de la Découverte. Nous allons voir maintenant comment ce programme peut être transformé en programme du Pendu.

Le programme du Pendu

Nous avons déjà appris à jouer au Pendu. Nous allons maintenant montrer comment obtenir un programme de Pendu en modifiant quelques sous-programmes du programme du Jeu de la Découverte. Les principales modifications sont présentées de la figure 2.13 à la figure 2.17. Analysons-les, rapidement :

Tout d'abord, notons que la structure principale du Pendu reste identique à celle du Jeu de la Découverte. Autrement dit, le programme de la figure 2.2, tel qu'il est, restera la structure principale du Pendu. Cela implique un léger compromis, puisque la commande « N », entrée par le joueur et identifiée comme telle à la ligne 170, n'aura aucune signification dans le jeu du Pendu. Ce

problème est réglé par une modification de la routine de paramétrage à la ligne 590 (routine appelée à partir de la ligne 110). Cette routine devra être modifiée pour ne pas tenir compte des demandes de changement de jeu.

En suivant la même progression que pour le Jeu de la Découverte, nous allons maintenant voir les parties de programme modifiées

```
# Effacer l'écran et rènvoyer le curseur à la maison (« home », en haut à gauche)

# Version Pet

effacecran  PRINT "clr";
            RETURN

     740  PRINT CHR$(147);:RETURN

# Version Apple

effacecran  HOME
            RETURN

     740  HOME:RETURN

# Version TRS-80

effacecran  CLS
            RETURN

     740  CLS:RETURN
```

Voici la routine qui efface l'écran et renvoie le curseur dans le coin supérieur gauche. On efface l'écran dans ce jeu pour deux raisons : pour d'une part n'avoir qu'une seule partie à la fois sur l'écran, et d'autre part pour dissimuler le mot secret après son entrée en mémoire.

La routine est présentée d'abord en BASIC Libre, puis en BASIC, pour le Pet, l'Apple et le TRS-80.

Figure 2.12 : Effacement de l'Ecran

pour le Pendu. La figure 2.13 montre la nouvelle forme des routines des figures 2.3 et 2.4. Pour le Pendu, la routine de création des nombres (lignes 310 à 350 de la figure 2.3) est supprimée puisqu'on ne se sert que de mots composés par l'un des joueurs. La version « Pendu » du programme conserve sous une forme similaire la routine qui va de la ligne 240 à 300 (fig. 2.4), en ce sens qu'elle saisit le mot à découvrir fourni par le premier joueur et l'encode dans le tableau HT. Il faut aussi que le premier joueur puisse entrer le nombre de réponses inexactes auquel aura droit le deuxième joueur.

Le contenu de la chaîne de caractères IN$ (message de rappel) est également différent de ce qu'il est dans le Jeu de la Découverte, parce que chaque essai dans le Jeu du Pendu est, soit un caractère unique, soit un mot complet. IP est initialisé à cet endroit du programme, car cette variable n'est pas utilisée de la même façon dans les deux jeux. Dans le Jeu de la Découverte, IP concerne exclusivement le coup en cours ; dans le Pendu, au contraire, une fois qu'un caractère est trouvé, il reste affiché jusqu'à la fin de la partie, aux positions concernées. En conséquence, IP part de zéro et croît avec le remplissage des places. L'initialisation de GL veut dire que la longueur des coups est de un caractère (un contrôle particulier a lieu par la suite pour traiter le cas où le joueur propose un mot complet).

Le tableau GS mentionné à la ligne 300 de la figure 2.13 est un nouveau-venu. Ce tableau sert à conserver les caractères trouvés par le joueur. Dans cette routine, GS est initialisé avec des tirets en nombre égal à celui des caractères du mot secret. Par la suite, les caractères trouvés iront remplacer les tirets.

La figure 2.14 montre le modèle « Pendu » de la routine de la figure 2.5. Les deux routines sont très semblables. L'une des deux différences consiste à afficher une ligne de la forme

```
– Y – Y – Y      VOTRE CREDIT D'ERREURS : 5
```

L'autre différence est que dans la version « Pendu », une réponse qui contient autant de caractères que le mot secret, est acceptée sans autre contrôle. Dans le cas contraire, les caractères du coup sont stockés dans la table GC en utilisant les mêmes instructions qu'à la figure 2.5. Bien sûr, avec le Pendu, les instructions de la boucle FOR ... NEXT ne sont exécutées qu'une seule fois, GL valant toujours 1. C'est ainsi que GC(1) sert à stocker le code ASCII de l'unique caractère du coup, et que le reste de la table GC est inutilisé.

La figure 2.15 montre la version « Pendu » de la routine de la figure 2.6. Cette nouvelle version commence par regarder si la pro-

```
# Entrer le mot caché — pour le Pendu

reflexion  GOSUB effacecran
           repeat {
               INPUT "MOT SECRET"; H$
               N = LEN(H$)
               } until (0 < N < = MX)
           INPUT "COMBIEN D'ERREURS PERMISES"; WG
           IN$ = "DEVINEZ UNE SEULE LETTRE OU TOUT LE MOT "
           GL = 1 : IP = 0                      # « IP, en place » part de
                                                zéro, ne décroît jamais
           MG = 99 : VG = 5                     # valeurs arbitraires
           GOSUB effacecran                     # dissimuler le mot secret
           FOR ZZ = 0 TO nombrascii             # mise à zéro table « coups »
               HT(ZZ) = 0
               NEXT ZZ
           FOR ZZ = 1 TO N
           HH = ASC(MID$(H$, ZZ, 1))        # remplissage table
                                            « coups »
               HT(HH) = HT(HH) + 2 ↑ (ZZ − 1)
               GS(ZZ) = ASC("−")                # « déjà trouvé » = tirets
                                                partout
               NEXT ZZ
           RETURN

200   GOSUB 740
210*  INPUT "MOT SECRET";H$:N=LEN(H$)
220   IF N=0 OR N>MX THEN 210
230*  INPUT "COMBIEN D'ERREURS PERMISES";WG
240   IN$="DEVINEZ UNE SEULE LETTRE OU TOUT LE MOT "
250   GL=1:IP=0:MG=99:VG=5:GOSUB 740
280** FOR ZZ=0 TO 255:HT(ZZ)=0:NEXT ZZ
290   FOR ZZ=1 TO N:HH=ASC(MID$(H$,ZZ,1))
300   GS(ZZ)=ASC("−")
310   HT(HH)=HT(HH)+2↑(ZZ−1):NEXT ZZ
320   RETURN
```

Voici la version « Pendu » des routines présentées aux figures 2.3 et 2.4. L'encodage du tableau des « coups » est identique. Un nouveau tableau appelé GS est initialisé avec des tirets partout. Les lettres trouvées viendront remplacer les tirets correspondants.

---

* La version Apple ne diffère que par la constante littérale.

** Sur Apple et TRS-80, remplacer 255 par 127.

Figure 2.13 : Le Pendu — Routine « réflexion »

```
# Saisie du coup joué

coup  PRINT                                         # à la ligne
      FOR ZZ = 1 TO N                               # afficher mot en partie trouvé
           PRINT CHR$(G$(ZZ) ;
           NEXT ZZ
      PRINT " VOTRE CREDIT D'ERREURS: "; WG         # ainsi que le crédit d'erreurs
      repeat {
           INPUT " VOTRE PROPOSITION "; GS
           IF LEN(G$) = N THEN
                RETURN
           else IF LEN(G$) < > GL THEN
                PRINT IN$
           } until (LEN(G$) = GL)
      FOR ZZ = 1 TO GL                              # mettre caractères coup dans
           GC(ZZ) = ASC(MID$(G$, ZZ, 1))            GC
           NEXT ZZ
      RETURN
```

```
360   PRINT:FOR ZZ=1 TO N:PRINT CHR$(G$(ZZ));:NEXT ZZ
365*  PRINT "VOTRE CRÉDIT D'ERREURS": WG
370*  INPUT "VOTRE PROPOSITION";G$
375   IF LEN(G$)=N THEN RETURN
380   IF LEN(G$)<>GL THEN PRINT IN$:GOTO 370
390   FOR ZZ=1 TO GL
395   GC(ZZ)=ASC(MID$(G$,ZZ,1)):NEXT ZZ
400   RETURN
```

Voici la version « Pendu » de la routine présentée à la figure 2.5. Les seules différences par rapport à la figure 2.5 sont : (1) l'affichage initial de la solution partielle et du crédit d'erreurs, et (2) le RETURN immédiat si LEN(G$)=N.

* La version Apple ne diffère que par la valeur des constantes littérales.

Figure 2.14 : Le Pendu — Routine « coup »

position formulée par le joueur, n'aurait pas le même nombre de caractères que la solution. Souvenons-nous que dans ce cas, la routine de la figure 2.14 l'a laissé passer sans le contrôler. Le sous-programme de la figure 2.15 compare directement cette réponse du joueur au mot à trouver stocké dans H$. S'il y a stricte égalité entre les deux, IP est forcé à N pour avertir que le joueur a trouvé la solution exacte. Dans le cas contraire, le coup est traité comme tous les autres coups erronés.

Si la réponse du joueur n'a pas le même nombre de caractères que celui du mot à trouver, le caractère unique sauvegardé en GC(1) sera comparé au contenu de l'élément correspondant du tableau HT. Cette opération a lieu dans la boucle FOR ... NEXT contrôlée par ZZ. (La boucle n'est exécutée qu'une fois, car GL vaut 1.) Le contenu HH de l'élément concerné de HT est analysé (comme en figure 2.6) pour connaître les puissances de deux qui le composent. Pour chacune des positions pour laquelle la puissance de deux correspondante est présente, le caractère trouvé est mis en place dans le tableau « résultats partiels » GS, en remplacement des tirets initiaux. Ceci se fait grâce à l'instruction

GS(YY) = GC(ZZ)

à la ligne 450.

Une erreur intéressante se produisait dans une version précédente de la routine de la figure 2.15. Le test de la ligne 410 était LEN(G$) = GL au lieu de LEN(G$) < > N. En principe, cela n'aurait dû avoir aucune influence puisque les deux seules valeurs que LEN(G$) peut présenter à l'entrée de cette routine, sont N et GL, de telle sorte que les deux tests auraient dû donner des résultats identiques. Néanmoins, il y a ambiguïté si N = GL. Dans ce cas particulier, la branche suivie ne devrait pas avoir d'importance, puisqu'une comparaison globale de la proposition du joueur devrait aboutir au même résultat qu'une analyse de chacun de ses caractères. Cependant, quand GL = N, la comparaison, caractère par caractère, est inopérante : la routine de la figure 2.14 ne remplissant pas le tableau GC quand LEN(G$) = N. Aussi a-t-il fallu coordonner le test du début de la routine de la figure 2.15 avec celui de la figure 2.14 (l'autre solution étant de modifier la routine de la figure 2.14 pour pouvoir la mettre « entre toutes les mains », et ceci en supprimant le test LEN(G$) = N dans le cas où l'on a GL = N).

On dit que deux sous-programmes ayant besoin d'être coordonnés de la sorte, sont « *couplés* ». Moins il y aura de couplage dans vos programmes, plus ils seront faciles à modifier et plus vos chances de ne pas découvrir un cas particulier d'erreur, bien après que les jeux d'essai aient été franchis avec succès, seront importantes.

```
    # Comparer le coup et le mot ou le nombre secret
comparer  IF LEN(G$) = N THEN
              IF G$ = H$ THEN IP = N                    # mot trouvé en entier
              else WG = WG - 1
          else {
              IX = 0                                    # compte présences du
                                                        car. dans mot
              FOR ZZ = 1 TO GL
                  HH = HT(GC(ZZ))                       # consultation table
                                                        « coups » pour car. proposé
                  IF HH <> 0 THEN
                      FOR YY = 1 TO N
                          IF HH est impair THEN {
                              IP = IP + 1               # un caractère de plus en
                                                        place
                              IX = IX + 1               # une place de plus occu-
                                                        pée par ce car.
                              GS(YY) = GC(ZZ)           # remplacer tiret par car.
                              }
                          HH = INT(HH/2)
                          NEXT YY
                  NEXT ZZ
              IF IX = 0 THEN
                  WG = WG - 1
              }
          RETURN

410 IF LEN(G$)<>N THEN 440
420 IF G$=H$ THEN IP=N:RETURN
430 WG=WG-1:RETURN
440 IX=0:FOR ZZ=1 TO GL:HH=HT(GC(ZZ))
445 IF HH=0 THEN 460
450 FOR YY=1TO N:IF HH<>2*INT(HH/2)THEN IP=IP+1:IX=IX+1:GS(YY)=GC(ZZ)
455 HH=INT(HH/2):NEXT YY
460 NEXT ZZ
465 IF IX=0 THEN WG=WG-1
470 RETURN
```

Voici la version « Pendu » de la Routine de la figure 2.6. Les routines sont semblables, mais avec de nombreuses petites différences. La variable RG n'existe plus, mais IP représente toujours le nombre « en place ». Si le mot complet proposé par le joueur est exact, cela est signalé en forçant IP à N.

Figure 2.15 : Le Pendu — Routine « comparer »

La figure 2.16 est la version « Pendu » de la routine de la figure 2.7. Les deux sous-programmes diffèrent sur deux points. Premièrement la nouvelle variable WG (wrong guesses, suppositions erronées) doit être testée, et deuxièmement, dans le Jeu du Pendu, on n'a pas à fournir au joueur des indications du genre

EXACTS : 3 EN PLACE : 1

La version « Pendu » de la routine de la figure 2.8 n'est pas donnée. Les deux versions sont en effet identiques, à ceci près que lorsque la routine de la figure 2.8 parle de la solution en utilisant l'article « L », le Pendu se sert de la variable H$ qui contient la solution. Par exemple, la ligne 340 de la version « Pendu » est
PRINT "VOUS AVEZ TROUVE"; H$; "EN"; G; "COUPS."
Une modification analogue a été faite à la ligne 570, faisant ainsi disparaître l'omission plutôt voyante du programme de la figure 2.8 : en effet, dans ce dernier, on ne dit jamais au joueur quelle est la bonne réponse.

```
# Annoncer gain ou perte partie

indication   IF IP = N THEN
                PRINT "C'EST EXACT !";
         else IF G = MG OR WG < 0 THEN {
                PRINT " COUPS TROP NOMBREUX — C'EST PERDU! "
                IP = - 1
                }
         RETURN


 480  IF IP=N THEN PRINT "C'EST EXACT!";:GOTO 500
 490  IF G=MG OR WG<0 THEN PRINT "COUPS TROP NOMBREUX—C'EST PERDU!":
      IP=-1
 500  RETURN
```

Voici la version « Pendu » de la routine présentée à la figure 2.7. La différence réside dans le fait que cette routine teste WG<0 aussi bien que G=MG pour terminer éventuellement la partie et ne donne par ailleurs aucune indication au joueur. C'est la routine de la figure 2.14 qui donne au joueur des renseignements sur sa façon de jouer.

Figure 2.16 : Le Pendu — Routine « indication »

```
# Paramétrage nouvelle partie

param   GOSUB effacecran
        RETURN

   590  GOSUB 740:RETURN
```

Voici la version « Pendu » de la routine présentée à la figure 2.9. On ne peut pas dire qu'elle soit surchargée de travail.

Figure 2.17 : Le Pendu — Routine « param »

La figure 2.17 montre la nouvelle version de la routine de la figure 2.9 qui se réduit maintenant à un simple appel à la routine d'effaçage d'écran. Toute complexité a disparu, étant donné qu'on ne propose plus au joueur de choisir parmi des variantes du jeu.

Il reste une modification à effectuer pour convertir le Jeu de la Découverte en Pendu : l'addition de GS(MX) à l'instruction de dimensionnement de la ligne 680 (voir figure 2.10). Le reste du programme du Jeu de la Découverte est repris sans modification dans le programme du Pendu.

Il est nécessaire d'étudier avec soin le processus de création du Pendu à partir du Jeu de la Découverte, car il illustre une technique très utile : la cannibalisation d'un programme en état de marche, dans le but d'en tirer rapidement un programme similaire. Alors que la présentation des différentes modifications a été longue et détaillée, la transformation effective du Jeu de la Découverte en Pendu a été effectuée en moins d'une heure. Lorsque vous évaluerez cette conversion, faites particulièrement attention à l'antagonisme entre simplicité de la conversion et clarté du programme résultant. Voyez s'il a été sage de conserver la variable GL (qui ne peut prendre que la valeur 1) ou la variable résiduelle L, qui, valorisée dans la routine d'initialisation, n'est par la suite jamais utilisée.

Additions et modifications possibles

Ce chapitre vient de présenter le Jeu de la Découverte (dix jeux en un seul) et le Pendu qui en est dérivé. On peut apporter de nom-

breuses améliorations à ces programmes. On trouvera ci-après plusieurs suggestions. Certaines sont faciles à programmer, tandis que pour d'autres, une préparation très détaillée sera indispensable.

- Enregistrer des statistiques. Par exemple, vous pourriez avoir envie de garder une trace du nombre de coups minimum, de leur moyenne (ou médiane), ou bien encore des résultats obtenus pour chacune des dix dernières parties.
- Pouvoir faire un retour en arrière au cours de la partie. Afficher, par exemple, un résumé de tous les coups précédemment joués et des commentaires de l'ordinateur. Utiliser un format d'affichage compact qui puisse figurer sur l'écran même dans le cas où le nombre de coups est exagérément élevé. Pour le Pendu, afficher un alphabet où chacune des lettres déjà éliminées, est remplacée par un astérisque.
- Revoir la façon d'afficher le dialogue et contrôler le curseur de façon à ce que le maximum d'informations reste sur l'écran.
- Donner au programme un soupçon d'intelligence. Par exemple, si au jeu de Quatre, un chiffre est proposé en une position dont l'impossibilité a été démontrée par un coup précédent, ou si, au Pendu, on utilise une lettre déjà éliminée, le programme pourrait poliment et gentiment, réprimander le joueur. (Par exemple, il pourrait dire VOUS AVEZ DEJA ESSAYE J, NE SERIEZ-VOUS PAS QUELQUE PEU DANS LES NUAGES ?). Ce genre de comportement de la part du programme doit être pensé avec soin, car toute manifestation prétentieuse de supériorité intellectuelle de la part du programme, risque de mettre le joueur hors de lui.
- Si l'ordinateur a une sonorisation, utiliser celle-ci pour réprimander ou féliciter le joueur.
- Mettre au point une méthode valable pour préciser les valeurs des niveaux VG (« Excellent ») et MG (« nombre maximum de coups »). Ces valeurs devraient tenir compte de la force du joueur.

En résumé

Nous avons commencé ce chapitre par une analyse des jeux de Découverte. Les règles du Jeu de Quatre ont été expliquées puis, nous en avons tiré celles de huit jeux de nombre et celles du jeu de mot, le Mot. Un exemple de partie de Quatre a été présenté pour illustrer les principes du jeu.

Un programme, appelé Jeu de la Découverte, a été développé pour réaliser Quatre et les neuf jeux qui lui sont apparentés. Son analyse a porté essentiellement sur sa structure et les techniques utilisées pour obtenir un programme suffisamment général qui prenne en compte la totalité des dix jeux.

Après avoir été étudié dans le détail, le programme du Jeu de la Découverte a alors été transformé en programme pour le Jeu du Pendu et ceci, par cannibalisation, c'est-à-dire par la modification ou le remplacement de quelques routines-clés. Cette transformation avait pour but de montrer les techniques qui permettent de convertir un programme existant en un autre programme similaire, cette façon de faire étant un bon moyen d'en réaliser rapidement de nouveaux.

Enfin, plusieurs suggestions ont été faites pour les améliorer. Le but de ces suggestions était d'une part, de montrer le genre d'esprit critique que l'on peut donner aux programmes et d'autre part, de proposer des exercices intéressants et enrichissants.

CHAPITRE **3**

Jeux horaires

Le Temps est un élément extrêmement important dans les jeux. La rapidité et le chronométrage de l'action sont souvent essentiels à leur intérêt. Comme nous le verrons, les techniques de contrôle de l'élément temps sont simples, une fois qu'elles ont été assimilées. Les programmes de ce chapitre ont été conçus pour illustrer l'utilisation de l'élément temps.

L'Horloge du Pet

Les jeux de ce chapitre ont tous été écrits pour le micro-ordinateur Pet, car celui-ci possède une horloge incorporée facile à utiliser. Voyons d'abord comment fonctionne cette horloge :

Dans la machine se trouve une horloge à quartz qui « bat » soixante fois par seconde. Le Pet transforme chaque « battement » en une chaîne de caractères de 6 chiffres stockés dans la variable TIME$. Les six chiffres indiquent les heures, les minutes et les secondes écoulées depuis minuit. Par exemple, la chaîne " 042715 " correspond à 4 h 27 min et 15 s. La chaîne de caractères " 120000 " correspond à midi et " 235959 " à une seconde avant minuit.

Malheureusement, le Pet n'a ni batterie ni autre source de courant qui maintienne l'horloge en fonction une fois l'alimentation principale coupée. Aussi, chaque fois que vous mettez le Pet en route, l'horloge est réglée à 000000 c'est-à-dire minuit. Heureusement, vous pouvez lui indiquer l'heure exacte en précisant la valeur de la variable TIME$ Par exemple, si vous venez de mettre le Pet sous tension et que l'heure exacte soit 10 h 27 du matin, il est possible d'entrer au clavier

TIME$ = " 102700 "

pour régler l'horloge. Pour le faire de façon précise, vous pouvez frapper

TIME$ = " 102800"

et attendre 10 h 28 exactement pour appuyer sur RETURN. L'horloge du PET une fois réglée, vous pourrez avoir l'heure en tapant (ou en insérant dans votre programme) l'instruction

PRINT TIME$

Le Pet affichera alors les six caractères donnant les heures, les minutes et les secondes selon le format indiqué plus-haut.

En ce qui concerne l'utilisation de l'horloge du Pet, c'est tout. (Il y a une autre variable TI, dans laquelle le Pet traite le temps sous un autre format, mais nous ne nous en servirons pas.) On peut avoir des horloges sur les ordinateurs Apple et TRS-80, et si votre

ordinateur en a une, vous pourrez utiliser les programmes présentés dans ce chapitre. Les parties de programmes qui concernent de façon spécifique l'horloge du Pet ont été rassemblées en groupes d'instructions nettement définis, groupes qui peuvent être facilement remplacés par les instructions équivalentes sur Apple et TRS-80.

L'Horloge

Commençons par un jeu qui n'en est pas réellement un : nous allons transformer le Pet en une horloge digitale (très coûteuse). Dans ce « jeu », l'ordinateur affiche de façon continue l'heure, en bas à droite de l'écran. Cet affichage se fait suivant le format courant de la plupart des horloges c'est-à-dire sur douze heures. Par exemple, si l'horloge du Pet indique comme heure 152537, notre affichage sur l'écran sera 3 h 25 min 37 s de l'après-midi. Pendant que notre horloge marque les secondes dans un coin de l'écran, elle attend également des ordres en provenance du clavier. (Malheureusement, on ne peut pas sur Apple attendre une entrée au clavier simultanément au déroulement du programme, car la version Apple de l'instruction GET attend la réponse, contrairement au GET du Pet ou à la

```
# Programme de l'Horloge
    GOSUB init
    GOSUB effacecran
    repeat {
        GOSUB evenements        # vérifier alarme, etc...
        GOSUB heurenplace       # curseur en position affichage heure
        GOSUB afficheure        # afficher l'heure
        GOSUB commande          # traitement commande, s'il y a lieu
        }

100  GOSUB 1110:GOSUB 920
110  GOSUB 340:GOSUB 240:GOSUB 250:GOSUB 120:GOTO 110
```

Voici la routine principale du programme de l'Horloge. Le programme affiche l'heure de façon continue en un point stable de l'écran. A chaque battement de l'horloge, le programme vérifie s'il n'y a pas lieu de déclencher certains événements qui doivent avoir lieu soit périodiquement soit à une heure fixée : affichage d'un message d'alarme et réglage de l'heure pour compenser l'imprécision du circuit de l'horloge. Le programme contrôle également, en permanence, les entrées de commandes faites à partir du clavier sous forme de caractères uniques. Les entrées peuvent suspendre l'affichage de l'heure en cas de dialogue, mais l'heure est entretenue automatiquement pendant ce dialogue.

Figure 3.1 : L'Horloge

fonction INKEY$ du TRS-80. Cependant, sur Apple, un signal en provenance du « manche à balai » peut servir à prévenir le programme que l'utilisateur désire entrer une commande au clavier).

Ces commandes comportent chacune un seul caractère. Leur signification figure ci-dessous :

F Avance l'horloge. Le programme demandera un nombre N. Une seconde sera ajoutée à l'heure toutes les N secondes.

S Retarde l'horloge. Le programme demandera un nombre N. Une seconde sera retranchée toutes les N secondes.

T Met à l'heure. Le programme demandera une chaîne de six chiffres (heures, minutes, secondes) conforme au format décrit précédemment. Cette chaîne servira au réglage de l'horloge.

A Règle l'heure d'alarme. Le programme demandera l'entrée d'une chaîne de six chiffres selon le format utilisé pour T. Quand l'horloge parviendra à l'heure indiquée, une alarme se déclenchera. (Le Pet étant un ordinateur silencieux, il s'agira d'une alarme visuelle.)

Q Coupe l'alarme (si elle s'est déjà déclenchée) et annule l'heure d'alarme de sorte que le lendemain, celle-ci ne se déclenchera pas.

R Coupe l'alarme, mais n'annule pas l'heure de déclenchement (en conséquence, le lendemain l'alarme se déclenchera à la même heure).

Z Change le fuseau horaire (ou avance ou retarde par rapport au soleil). Le programme demande le nombre d'heures du décalage. La réponse doit être un nombre entier positif N pour avancer l'horloge de N heures, ou un nombre entier négatif – N pour la retarder de N heures.

C Modifie l'heure de quelques secondes. Le programme demande le nombre de secondes. La réponse sera un nombre entier positif N pour avancer l'heure de N secondes, ou un nombre entier négatif – N pour la retarder de N secondes.

Il va de soi que ces commandes ne représentent qu'un sous-ensemble des possibilités que l'on pourrait avoir avec une horloge contrôlée par ordinateur.

Le Programme de l'Horloge

La figure 3.1 montre la routine principale. Les programmes concernant l'horloge seront donnés en « BASIC Libre » et en BASIC,

mais aucune référence ne sera faite dans le texte à la version du « BASIC Libre ». Si vous avez regardé les versions « BASIC Libre » des précédents programmes et essayé de les comprendre (ou si vous avez lu au chapitre 6 la description du « BASIC Libre ») vous vous êtes certainement déjà rendu compte que le « BASIC Libre » est plus facile à suivre que le BASIC proprement dit.

Le programme « Horloge » fait essentiellement appel au sous-programme d'initialisation et à celui qui efface l'écran, puis se poursuit par une boucle sans fin. Dans cette boucle, le temps est affiché de façon continue, les entrées au clavier sont traitées, et les heures calculées pour les événements prévus sont comparées au temps instantané. La mise à jour de l'heure réelle se fait de façon automatique dans le Pet, de sorte qu'aucun retard ne se produit quand le programme traite une commande entrée au clavier. Tout événement (par exemple l'alarme qui « retentit ») qui voit son heure d'exécution passer pendant le traitement d'une commande, se produira dès que le traitement en cours sera terminé. Nous verrons plus tard comment cela se fait, lorsque nous commenterons la figure 3.4.

La figure 3.2 montre le sous-programme de traitement des commandes qui est appelé à partir de la boucle principale de la figure 3.1. Ce que fait ce sous-programme est très facile à comprendre. Si l'on n'a appuyé sur aucune touche, le sous-programme fait simplement retour (c'est-à-dire qu'il exécute l'instruction RETURN). Si par contre, une touche a été enfoncée, le sous-programme efface l'écran appelle le sous-programme adéquat pour le traitement de la commande, efface à nouveau l'écran et fait retour.

La figure 3.3 montre les deux sous-programmes impliqués dans l'affichage de l'heure. Le premier ne fait que placer le curseur à l'endroit convenable sur l'écran. Le second a deux fonctions distinctes : (1) il convertit le format de la chaîne de caractères donnant le temps en heures, minutes et secondes, en format sur 12 h (du matin ou de l'après-midi) et (2) affiche l'heure d'après ce nouveau format.

La réunion de ces deux fonctions distinctes en un seul sous-programme n'a rien d'idéal du point de vue de la conception du programme. La séparation de la fonction de conversion et de la fonction d'affichage rendrait plus aisé le développement ultérieur. C'est ainsi que l'une des routines pourrait faire passer le temps du format TIME$ au format matin/après-midi tandis qu'une deuxième en assurerait l'affichage effectif. Ceci permettrait de faire appeler la routine de conversion par celle de mise à l'heure (fig. 3.6) pour montrer à l'utilisateur l'heure exacte sur laquelle il doit régler l'horloge. D'autres utilisations de la routine de changement de format peuvent

être imaginées. (A titre d'exercice, entreprenez la séparation des deux sous-programmes en question.)

La figure 3.4 montre la routine de contrôle des événements. Pour comprendre ce sous-programme, il faut saisir la signification du temps « Julien » utilisé pour la prévision des événements. A chaque instant, le temps Julien est le nombre de secondes écoulé depuis le dernier minuit. Si par exemple, il est 10 h du soir et que vous

```
# Traitement des commandes
commande   GET CM$
           IF CM$ <> "" THEN {
                GOSUB effacecran
                on case CM$ GOSUB
                     "F"   avancer
                     "S"   retarder
                     "T"   misalheure
                     "A"   reglalarme
                     "Q"   stopalarme
                     "R"   reparalarme
                     "Z"   heuremodif
                     "C"   secmodif
                GOSUB effacecran
                }
           RETURN
```

```
120* GET CM$:IF CM$=""THEN 230
130  GOSUB 920
140  IF CM$="F"THEN GOSUB 450:GOTO 220
150  IF CM$="S"THEN GOSUB 480:GOTO 220
160  IF CM$="T"THEN GOSUB 520:GOTO 220
170  IF CM$="A"THEN GOSUB 550:GOTO 220
180  IF CM$="Q"THEN GOSUB 590:GOTO 220
190  IF CM$="R"THEN GOSUB 600:GOTO 220
200  IF CM$="Z"THEN GOSUB 690:GOTO 220
210  IF CM$="C"THEN GOSUB 720:GOTO 220
220  GOSUB 920
230  RETURN
```

Cette routine, qui est appelée de façon continue dans la boucle de la figure 3.1, cherche les commandes « mono-caractères » entrées au clavier. Si l'une d'entre elles a été entrée la routine correspondante est appelée.

* Sur TRS-80 il convient de remplacer GET CM$ par CM$=INKEY$.

Figure 3.2 : L'Horloge — Traitement des commandes

désirez régler le déclenchement de l'alarme pour 1 h du matin, le temps Julien de déclenchement de l'alarme sera de 90 000, car c'est le nombre de secondes qui s'écoule pendant les 25 h séparant le dernier passage à minuit (il y a de cela 22 h) et le moment choisi pour l'alarme (dans 3 h). Quand arrivera minuit, deux heures après que l'heure d'alarme ait été réglée, l'origine du temps Julien changera. A cet instant l'actionnement de l'alarme doit être réglé à 3600 car c'est le nombre des secondes comprises dans l'heure qui sépare le

```
# Mettre curseur à position affichage heure

heurenplace  LL = TL : CC = TC              # ligne et colonne de l'heure
             GOSUB curseur
             RETURN

# afficher l'heure

afficheure   GOSUB hms                      # obtenir heures, minutes, secondes
             HR = VAL(HR$)
             IF HR > 11 THEN
                  AP$ = " PM"
             else
                  AP$ = " AM"
             IF HR = 0 THEN
                  HR$ = " 12"
             else IF HR > 12 THEN
                  HR$ = STR$(HR - 12)
             else
                  HR$ = STR$(HR)
             HR$ = RIGHT$ (" "+ HR$, 2)
             PRINT HR$ ; " : " ; MN$ ; " : " ; SC$ ; " " ; AP$ ;
             RETURN

240 LL=TL:CC=TC:GOSUB 930:RETURN
250 GOSUB 1050:HR=VAL(HR$)
260 IF HR>11 THEN AP$="PM":GOTO 280
270 AP$="AM"
280 IF HR=0 THEN HR$="12":GOTO 310
290 IF HR>12 THEN HR$=STR$(HR-12):GOTO 310
300 HR$=STR$(HR)
310 HR$=RIGHT$(" "+HR$,2)
320 PRINT HR$;":";MN$;":";SC$;" ";AP$;
330 RETURN
```

Ces routines affichent l'heure selon le format "HH:MM:SS AP" à un endroit bien défini de l'écran. L'heure est cadrée à droite avec un espace en première position pour les heures allant de 1 à 9.

Figure 3.3 : L'Horloge — Affichage de l'heure

```
# Contrôle de l'alarme, du réglage et des autres événements

evenements  GOSUB julien                  # obtenir heure « julienne »

            ET = JT

            IF ET < OE THEN {             # on vient de passer minuit
                 IF AJ <> - 1 THEN
                      AJ = AJ - secparjour
                 IF AL <> - 1 THEN
                      AL = AL - secparjour

            IF OE <> ET THEN {            # contrôler événements une fois par
                                          battement
                 OE = ET
                 IF AL <> - 1 AND AL < = ET THEN {
                      GOSUB declenche     # déclencher l'alarme
                      AL = AL + secparjour  # régler alarme pour demain même
                                          heure
                      }
                 IF AJ <> - 1 AND AJ < = ET THEN {
                      DT = AI             # modifier heure de AI secondes
                      GOSUB coupdepouce
                      AJ = AJ + AF        # fixer heure prochain ajustement
                      }
                 }
            RETURN
```

```
340  GOSUB 830:ET=JT
350  IF ET>=OE THEN 380
360  IF AJ<>-1 THEN AJ=AJ-86400
370  IF AL<>-1 THEN AL=AL-86400
380  IF OE=ET THEN 440
390  OE=ET
400  IF AL=-1 OR AL>ET THEN 420
410  GOSUB 610:AL=AL+86400
420  IF AJ=-1 OR AJ>ET THEN 440
430  DT=AI:GOSUB 810:AJ=AJ+AF
440  RETURN
```

Cette routine regarde si le moment d'un réglage de l'horloge ou d'un affichage de l'alarme est arrivé : les échéances sont notées en un temps dit « Julien » qui leur permet (en principe) d'être de plusieurs jours au-delà de l'heure présente.

Figure 3.4 : L'Horloge — Contrôle des événements

minuit le plus récent, de l'instant toujours prévu pour le déclenchement de l'alarme.

Chaque fois que l'horloge passe minuit, la routine de contrôle des événements effectue cette modification de l'origine du temps en soustrayant des heures prévues pour chaque événement, le nombre 86 400 (nombre de secondes dans une journée).

L'utilisation de ce temps « Julien » permet au programme de faire la distinction entre deux moments séparés par 24 h. Par exemple, supposons que l'alarme doive se déclencher à 9 h du matin et qu'à 8 h 59 vous décidiez de vous servir d'une des commandes. Peut-être avez-vous remarqué que l'horloge prenait un peu de retard, et vous avez utilisé la commande F pour l'accélérer. Mais que se passe-t-il s'il vous faut un petit moment pour décider de la bonne période à entrer ? : au moment où vous avez terminé, il est 9 h 01. La routine de contrôle des événements est alors appelée, et s'apercevant que l'heure du moment (32 460 pour 9 h 01 du matin) a dépassé l'heure de l'alarme (32 400 pour 9 h du matin) elle déclenche alors l'alarme. La routine réalise sans difficultés que l'heure de l'alarme est 9 h 00 du matin aujourd'hui et non demain, car si c'était demain, l'heure de l'alarme serait réglée au temps Julien 118 800.

Pour que cette procédure fonctionne, la routine d'événements doit pouvoir reconnaitre que minuit est déjà passé. C'est là le rôle premier de la variable OE (Probablement comme Old Events : Evénement passés. *N.d.T.*) qui garde en mémoire l'heure du dernier contrôle d'événement effectué. Si la routine d'événements constate que le temps réel Julien est inférieur à celui du dernier contrôle d'événement, elle en « déduit » que minuit vient juste de passer.

L'autre raison pour garder trace du dernier contrôle est qu'il faut s'assurer que le contrôle d'événement n'est effectué qu'une seule fois par battement d'horloge. Cette assurance peut être précieuse lors de l'addition de fonctions supplémentaires mais n'est pas nécessaire dans la version actuelle du programme, car, la suite donnée aux événements est réalisée de telle façon qu'il n'y aurait pas d'inconvénients à ce que le contrôle eût lieu plus d'une fois par seconde.

La routine d'événements contrôle la réalisation de trois événements :

(1) le passage de minuit, (2) il y a lieu de faire « retentir » l'alarme, et (3) il convient de procéder à un réglage.

Lorsque l'heure de l'alarme arrive, celle-ci « retentit » et le nombre de secondes existant dans une journée est ajouté à la valeur de la variable AL qui contient l'heure de l'alarme. C'est-à-dire que l'alarme est automatiquement réglée pour se déclencher le lendemain à la même heure. Si l'heure est venue de faire un réglage d'horloge,

celle-ci est effectuée et l'heure du prochain réglage est fixée. Il faut noter que ce dernier est calculé en ajoutant la période de réglage à l'heure prévue et non à l'heure où celui-ci est effectivement réalisé. Ce qui signifie que si ce réglage est fait avec une minute de retard, par exemple si le programme traite une autre commande, le prochain ajustement sera prévu pour la même heure que s'il venait d'avoir lieu. Qui plus est, cela signifie que si plusieurs réglages ont été sautés

```
# Rendre horloge plus rapide

avancer      repeat
                INPUT "INTERVALLE DE REGLAGE(SEC) "; FT
                until (FT > 0 et entier)
             IN = 1
             GOSUB parareglage
             RETURN

# Rendre horloge plus lente

retarder     repeat
                INPUT " INTERVALLE DE REGLAGE (SEC) "; FT
                until (FT > 0 et entier)
             IN = - 1
             GOSUB parareglage
             RETURN

# Préciser paramètres réglage

parareglage  AI = IN                      # fixer valeur incrément
             AF = FT                      # et période (intervalle)
             GOSUB julien
             AJ = JT + AF
             RETURN

450  INPUT "INTERVALLE DE REGLAGE (SEC)";FT
460  IF FT<=0 OR FT<> INT(FT) THEN 450
470  IN=1:GOSUB 510:RETURN
480  INPUT "INTERVALLE DE REGLAGE (SEC)";FT
490  IF FT<=0 OR FT<>INT(FT) THEN 480
500  IN=-1:GOSUB 510:RETURN
510  AI=IN:AF=FT:GOSUB 830:AJ=JT+AF:RETURN
```

Ces routines traitent les commandes F (« faster », plus rapide) et S (« slower », plus lent) qui règlent la vitesse de l'horloge. L'utilisateur peut demander qu'une seconde soit ajoutée ou retranchée à l'heure à un intervalle de temps déterminé (par exemple, toutes les 10 000 secondes). Cela permet de compenser l'inexactitude d'un circuit d'horloge par ailleurs précis.

Figure 3.5 : L'Horloge — Commandes de réglage de la vitesse

(soit parce que le programme a été occupé pendant longtemps à traiter une commande, soit parce qu'ils sont très fréquents), ils seront tous effectués (à raison d'un par seconde) dès que le programme aura enfin les « mains » libres.

La figure 3.5 montre les sous-programmes de traitement des commandes F et S. Ces routines calculent la période de réglage (AF) (Adjustment Frequency) et l'incrément (AI) qui seront utilisés par la routine de contrôle des événements ; elles précisent ensuite le moment (AJ) du premier réglage.

La figure 3.6 montre la routine de traitement de la commande T. Il faut noter que l'on demande à l'opérateur d'appuyer sur RETURN pour enclencher l'horloge. Vous en connaissez la raison si vous avez déjà essayé de mettre à l'heure l'horloge du Pet à l'aide d'une instruction de la forme

TIME$ = 120430

Après avoir attendu exactement qu'il soit 4 min et 30 s de l'après-midi, vous appuyez triomphalement sur RETURN, pour obtenir cette simple réponse de la part du Pet :

TYPE MISMATCH ERROR ?

ceci parce que vous avez oublié de placer la chaîne de caractères entre guillemets. La procédure utilisée à la figure 3.6 vous oblige

```
#  Mise à l'heure

misalheure   PRINT " MISE A L'HEURE "
             GOSUB quellheure
             PRINT " APPUYEZ SUR RETURN POUR DEMARRER HORLOGE "
             GOSUB uncar
             GOSUB nouvelheure          # TM$ devient la nouvelle heure
             RETURN

520  PRINT "MISE A L'HEURE"
530  GOSUB 640:PRINT "APPUYEZ SUR RETURN POUR DEMARRER HORLOGE"
540  GOSUB 1030:GOSUB 1080:RETURN
```

Voici la routine qui traite la commande T (« set time », mettre à l'heure). Le programme demande qu'on appuye sur la touche RETURN une deuxième fois pour faire repartir l'horloge, au lieu d'utiliser le RETURN qui termine la saisie de la chaîne de caractères contenant l'heure. Cette façon de procéder met l'utilisateur à l'abri de la déception qui résulterait de la découverte d'une erreur de syntaxe lorsque ce n'est plus l'heure.

Figure 3.6 : L'Horloge — Mise à l'heure

Fixer heure alarme

```
reglalarm     PRINT " HEURE D'ALARME "
              GOSUB quellheure
              AL = TM
              GOSUB julien
              IF AL < JT THEN
                  AL = AL + secparjour
              RETURN
```

Oter l'alarme

```
stopalarm     AL = - 1
              GOSUB couper
              RETURN
```

Reporter l'alarme

```
reporalarme   GOSUB couper
              RETURN
```

```
550  PRINT "HEURE D'ALARME"
560  GOSUB 640:AL=TM:GOSUB 830
570  IF AL<JT THEN AL=AL+86400
580  RETURN
590  AL=-1:GOSUB 630:RETURN
600  GOSUB 630:RETURN
```

Ces trois routines traitent les commandes ayant trait à l'alarme : A (fixer Alarme), Q (couper, Quitter l'alarme), R (effacer l'alarme et la Reporter). La routine qui fixe l'heure, saisit celle-ci suivant un format « 24 heures », et retient comme heure de déclenchement de l'alarme la prochaine occurrence de l'heure saisie. La commande Q force AL à −1, signalant ainsi au programme qu'il n'y a plus d'alarme en vigueur. La commande R ne change pas la valeur de AL. Dans la routine de la figure 3.4, AL est réglée automatiquement de façon à ce que l'alarme soit redéclenchée 24 heures plus tard. Le mécanisme de fonctionnement de AL pourrait prendre en compte des délais d'alarme supérieurs à 24 heures, mais aucune commande n'a été prévue pour cela.

Figure 3.7 : L'Horloge — Commandes de l'alarme

à appuyer sur RETURN une deuxième fois, cela fera démarrer l'horloge à coup sûr et vous n'aurez pas de surprises.

La figure 3.7 montre les routines qui traitent les commandes A, Q et R concernant l'alarme. Ces sous-programmes sont tout à fait rudimentaires, par conséquent, les possibilités d'alarme de ce programme d'horloge, sont très réduites. Plusieurs améliorations sont proposées à la fin de ce chapitre.

La figure 3.8 présente les routines chargées de l'affichage de l'alarme. Comme vous pouvez le voir, la place ne manque pas pour apporter des améliorations.

La figure 3.9 montre le sous-programme qui demande à l'utilisateur d'entrer l'heure (soit pour régler l'horloge, soit pour préciser l'heure d'alarme). Cette routine demande une entrée respectant le même format que celui qui est demandé par le Pet pour rentrer TIME$, bien qu'elle soit suffisamment évoluée pour accepter la chaîne de caractères donnant l'heure, avec ou sans guillemets.

Les figures 3.7, 3.8 et 3.9 fournissent des exemples de « souches », technique qui vient de l'utilisation de la méthode appelée « démarche descendante » pour la réalisation des programmes. La démarche descendante implique que la structure principale du programme soit précisée en premier (exemple à la figure 3.1). Dès

```
# Déclencher l' alarme

declencher  LL = AR : CC = AC        # ligne et colonne alarme
            GOSUB curseur
            PRINT AL$;               # afficher chaîne alarme
            RETURN

# Couper l'alarme

couper      GOSUB effacecran
            RETURN

610  LL=AR:CC=AC:GOSUB 930
620  PRINT AL$;:RETURN
630  GOSUB 920:RETURN
```

Ces routines ont trait à l'affichage de l'alarme. La première affiche une chaîne prédéfinie AL$ à un emplacement préétabli sur l'écran. La seconde efface l'écran. Un affichage variable (par exemple clignotant ou si une sonorisation existe émettant des « bips ») nécessiterait une approche tout à fait différente et mettrait en cause la routine de contrôle des événements de la figure 3.4.

Figure 3.8 : L'Horloge — Affichage de l'alarme

lors, chacun des sous-programmes qui forment les blocs de la structure principale, est à son tour détaillé et de façon analogue divisé en autres sous-programmes. (Par exemple, les figures 3.2, 3.3 et 3.4 montrent les principaux sous-programmes appelés par la routine principale de la figure 3.1.) Si à un moment quelconque de ce processus, il n'est pas possible d'écrire un des sous-programmes prévus, on le remplace par une « souche ». Une « souche » est une routine qui fera quelque chose de plausible quand on l'appellera, soit en réalisant l'action prévue mais sous une forme élémentaire, soit en faisant quelque chose qui soit similaire (mais plus simple). Par exemple, la routine d'affichage de l'heure à la ligne 250 (de la figure 3.3) pourrait en un premier temps, être représentée par la « souche » suivante :

```
PRINT "10 : 25 : 04 AM "
RETURN
```

Ou bien c'est la routine de traitement des commandes de la figure 3.2 qui pourrait exister d'abord dans une version « souche » qui ne traiterait que la commande T.

```
# Demander l'heure

quellheure   repeat {
                 INPUT " HEURE (6 CHIFFRES : HHMMSS) "; TM$
                 AH = VAL(LEFT$(TM$, 2))
                 AM = VAL(MID$(TM$, 3, 2))
                 AS = VAL(RIGHT$(TM$, 2))
                 } until (0 < = AH < 24 AND 0 < = AM < 60 AND 0 < = AS < 60)
             TM = 3600 * AH + 60 * AM + AS
             RETURN

640  INPUT «HEURE (6 CHIFFRES:HHMMSS)» TMS
650  AH=VAL(LEFT$(TM$,2)):IF AH<0 OR AH>=24 THEN 640
660  AM=VAL(MID$(TM$,3,2)):IF AM<0 OR AM>=60 THEN 640
670  AS=VAL(RIGHT$(TM$,2))IF AS<0 OR AS>=60 THEN 640
680  TM=3600*AH+60*AM+AS:RETURN
```

Cette routine permet à l'utilisateur de préciser l'heure de l'horloge ou celle de l'alarme. Le format est le même que celui qui permet de rentrer TIME$ ce qui montre que la routine a été écrite avec comme but de faciliter la tâche de programmeur et non celle de l'utilisateur.

Figure 3.9 : L'Horloge — Inélégante saisie de l'heure

Le mot « souche » a été introduit dans le vocabulaire pour désigner ces routines car la structure d'un programme développé en utilisant la démarche descendante, ressemble aux arbres inversés utilisés pour dessiner les organigrammes de nombreuses entreprises. Une souche se trouve souvent à un niveau hiérarchique élevé dans la structure du programme, de telle sorte que la description correspondante ne va pas plus loin, comme si le tronc d'un arbre inversé avait été coupé, ne laissant que la souche.

La technique des « souches » présente un inconvénient. En effet, il arrive qu'un sous-programme « souche » tienne finalement la place de la branche tout entière. C'est-à-dire que les caractéristiques et les fonctions qui devaient lui être ajoutées ultérieurement, n'ont jamais été effectivement programmées.

```
# Changer les heures

heuremodif   repeat
                  INPUT " VALEUR CHANGEMENT (HEURES) "; CH
                  until (CH soit un nombre entier AND - 24 < CH < 24)
             CS = CH * 3600                 # changement en secondes
             GOSUB modifhorloge
             RETURN

# Changer les secondes

secmodif     repeat
                  INPUT " VALEUR CHANGEMENT (SECONDES) "; CS
                  until (CS soit un nombre entier AND - 60 < CS < 60)
             GOSUB modifhorloge
             RETURN
```

```
690 INPUT"VALEUR CHANGEMENT (HEURES)"
700 IF CH<>INT(CH) OR CH<=-24 OR CH>=24 THEN 690
710 CS=CH*3600:GOSUB 750:RETURN
720 INPUT"VALEUR CHANGEMENT (SECONDES)";CS
730 IF CS<>INT(CS) OR CS<=-60 OR CS>=60 THEN 720
740 GOSUB 750:RETURN
```

Ces routines traitent la commande C (pour « changer l'heure d'un certain nombre de secondes ») et Z (pour « changer l'heure d'un certain nombre d'heures »). Chacune de ces routines exprime la modification demandée et appelle la routine de modification de l'heure de la figure 3.11.

Figure 3.10 : L'Horloge — Modification de l'heure

La figure 3.10 montre les routines de traitement des commandes C et Z, commandes qui permettent de modifier l'heure d'un nombre de secondes ou d'heures donné sans arrêter l'horloge. (Une routine modifiant l'heure d'un nombre donné de minutes, aurait pu également être fournie, mais son intérêt a paru insuffisant.)

La figure 3.11 montre la routine appelée par les deux routines de la figure 3.10. Le but de cette routine est de réaliser le réglage effectif de l'horloge en modifiant les variables AL et AJ dans le cas où la modification d'heure conduit au franchissement de minuit dans un sens ou dans l'autre. Deux autres niveaux de sous-pro-

```
# Modifier indications horloge
modifhorloge  GOSUB julien                        # noter heure avant changement
              CT = JT
              DT = CS                             # valeur changement
              GOSUB coupdepouce                   # le faire
              GOSUB julien                        # a-t-on franchi minuit
              IF CS < 0 AND JT > CT THEN {        # franchi minuit en arrière
                  IF AJ <> - 1THEN
                      AJ = AJ + secparjour
                  IF AL <> - 1 THEN
                      AL = AL + secparjour
                  }
              IF CS > 0 AND JT < CT THEN {        # franchi minuit en avant
                  IF AJ <> - 1 THEN
                      AJ = AJ - secparjour
                  IF AL <> - 1 THEN
                      AL = AL - secparjour
                  }
              RETURN
```

```
750  GOSUB 830:CT=JT
760  DT=CS:GOSUB 810:GOSUB 830
770  IF CS>=0 OR JT<=CT THEN 785
775  IF AJ<>-1 THEN AJ=AJ+86400
780  IF AL<>-1 THEN AL=AL+86400
785  IF CS<=0 OR JT>=CT THEN 800
790  IF AJ<>-1 THEN AJ=AJ-86400
795  IF AL<>-1 THEN AL=AL-86400
800  RETURN
```

Cette routine modifie les indications de l'horloge et s'assure que les variables AL et AJ sont remises à la valeur correcte si la modification fait franchir minuit.

Figure 3.11 : L'Horloge — Modifications de l'heure *(Précautions particulières)*

grammes concernent également l'horloge du Pet. Au premier niveau, montré à la figure 3.12, est réalisé l'algorithme qui permet d'effectuer les changements d'heure « au vol ». Au niveau inférieur (montré à la figure 3.15) se trouvent les routines qui s'occupent de façon spécifique de TIME$. Cette séparation des fonctions rend le programme Horloge facilement transposable aux autres systèmes possédant des horloges.

Le programme présenté à la figure 3.12 modifie le temps d'une quantité donnée (positive ou négative) précisée dans la variable DT. Le problème est que l'horloge bat la seconde. Si l'horloge bat entre l'instant où le programme a en « main » l'heure fournie par TIME$ et le moment auquel il met à jour TIME$, une seconde sera perdue. (Voyez-vous pourquoi ? Ceci est un exemple d'un problème qui intervient de façon caractéristique lorsqu'on doit traiter des processus simultanés.) La technique utilisée à la figure 3.12 consiste à prendre l'heure en cours dans TIME$, puis à attendre la prochaine seconde (en l'ajoutant dans le calcul de l'incrément) avant de mettre à jour TIME$. Cette méthode nécessite que tous les calculs durent moins d'une seconde, de façon à ce que la mise à jour de TIME$ ait lieu avant le deuxième battement suivant l'instant origine où a été saisie l'heure en cours.

La figure 3.13 montre les routines qui assurent le passage du format numérique « Julien » de l'heure, au format sous forme de

```
#  Modifier l'heure de DT secondes

coupdepouce  GOSUB julien        # calcul de JT à partir de l'ancienne heure
                                   (TT$)
             JT = JT + DT + 1    # additionner, plus un pour prochain batte-
                                   ment
             GOSUB nonjulien     # calcul nouvelle heure TM$ à partir de JT
             GOSUB nouveautic    # attendre battement suivant TT$
             GOSUB nouvelheure   # régler l'heure sur TM$
             RETURN

   810   GOSUB 830:JT=JT+DT+1:GOSUB 850
   820   GOSUB 1090:GOSUB 1080:RETURN
```

Cette routine ajoute algébriquement à l'heure un nombre donné de secondes DT. Pour éviter une erreur subtile, la routine attend systématiquement le battement suivant de l'horloge pour opérer la modification.

Figure 3.12 : L'Horloge — Technique de modification de l'heure

```
# Calculer le temps « Julien »

julien      GOSUB hms
            JT = VAL(SC$) + 60 * VAL(MN$) + 3600 * VAL(HR$)
            RETURN


# Passage du temps julien (JT) au format caractères (TM$)

nonjulien   repeat                        # modifier JT pour que 0 < = JT < secparjour
                IF JT < 0 THEN
                    JT = JT + secparjour
                until (JT > = 0)
            repeat
                IF JT > = secparjour THEN
                    JT = JT - secparjour
                until (JT < secparjour)
            SC = JT mod 60                                # extraire les secondes
            MN = (JT - SC)/60 mod 60                      # les minutes
            HR = (JT - SC - 60 * MN)/3600                 # puis les heures
            TX = SC + 100 * MN + 10000 * HR + 1000000     # concaténer avec un
                                                          1 à gauche
            TM$ = RIGHT$(STR$(TX), 6)                     # pour être sur d'avoir
                                                          des zéros à gauche
            RETURN
```

```
830 GOSUB 1050
840 JT=VAL(SC$)+60*VAL(MN$)+3600*VAL(HR$):RETURN
850 IF JT<0 THEN JT=JT+86400:GOTO 850
860 IF JT>=86400 THEN JT=JT-86400:GOTO 860
870 SC=JT-60*INT(JT/60)
880 MN=(JT-SC)/60:MN=MN-60*INT(MN60)
890 HR=(JT-SC-60*MN)/3600
900 TX=SC+100*MN+10000*HR+1000000
910 TM$=RIGHT$(STR$(TX),6):RETURN
```

Ces routines assurent l'aller et retour entre le format « littéral » de TIME$ et le format « julien » utilisé par la routine de contrôle des événements. L'astuce consistant à ajouter 1 000 000 à un nombre de six chiffres (ligne 900) de façon à être assuré d'avoir des zéros à gauche, mérite d'être retenue.

Figure 3.13 : L'Horloge — Routines pour le temps « Julien »

```
#  Effacer l'écran
effacecran   PRINT "clr"; : RETURN

#  Positionner le curseur sur la ligne (0 < = LL < 24) et la colonne (0 < = CC < 40)
curseur      LL = LL mod 24 : CC = CC mod 40
             PRINT "coin supérieur gauche";
             IF LL < > 0 THEN
                  FOR XX = 1 TO LL
                       PRINT "curseur en bas";
                       NEXT XX
             IF CC < > 0 THEN
                  FOR XX = 1 TO CC
                       PRINT "curseur à droite";
                       NEXT XX
             RETURN

#  Entrer un caractère unique dans X$
uncar        repeat GET X$ until (X$ < > "") : RETURN
```

```
 920* PRINT CHR$(147);:RETURN
 930* IF LL<0 THEN LL=LL+24:GOTO 930
 940* IF LL>23 THEN LL=LL-24:GOTO 940
 950* IF CC<0 THEN CC=CC+40:GOTO 950
 960* IF CC>39 THEN CC=CC-40:GOTO 960
 970* PRINT CHR$(19);
 980* IF LL=0 THEN 1000
 990* FOR XX=1 TO LL:PRINT CHR$(17);:NEXT XX
1000  IF CC=0 THEN 1020
1010* FOR XX=1 TO CC:PRINT CHR$(29);:NEXT XX
1020  RETURN
1030* GET X$:IF X$=""THEN 1030
1040  RETURN
```

Ces routines sont écrites pour le PET. L'effacement d'écran, le positionnement du curseur, et l'entrée d'un caractère unique sont réalisés différemment sur les autres systèmes.

---

* Sur TRS-80 l'effacement d'écran est obtenu par la commande CLS (voir figure 2.11) ; les numéros de lignes et de colonnes sont 16 et 64 plutôt que 24 et 40 ; « coin supérieur droit », « curseur en bas » et « curseur à droite » sont réalisés par CHR$(28), CHR$(26) et CHR$(25) ; le GET X$ est remplacé par X$=INKEY$ (voir figures 2.11 et 2.12).

Figure 3.14 : L'Horloge — Routines utilitaires *(pour le Pet)*

chaîne de caractères de TIME$ (et vice versa). Ces sous-programmes sont simples; néanmoins, il convient de s'assurer que le format « chaîne de caractères » possède, si nécessaire, des zéros à gauche. Le résultat final devant avoir six chiffres, on ajoute 1 000 000 au nombre, en ne gardant finalement que les six chiffres à droite du résultat.

Il faut noter que si le format de TIME$ est utilisé tout au long du programme, on ne se sert de la variable TIME$ elle-même, que dans les routines de la figure 3.15. Il ne serait pas nécessaire de changer

```
#  Aller chercher l'heure

hms          TT$ = TIME$                            #  la noter
             HR$ = LEFT$(TT$, 2)
             MN$ = MID$(TT$, 3, 2)
             SC$ = RIGHT$(TT$, 2)
             RETURN

#  Nouvelle heure est TM$

nouvelheure  TIME$ = TM$
             RETURN

#  Attendre prochain battement après TT$

nouveautic   repeat { } until (TT$ <> TIME$)
             RETURN
```

```
1050*  TT$=TIME$
1060*  HR$=LEFT$(TT$,2):MN$=MID  (TT$,3,2):SC$=RIGHT$(TT$,2)
1070   RETURN
1080** TIME$=TM$:RETURN
1090   IF TT$=TIME$ THEN 1090
1100   RETURN
```

Ces routines sont les seules du programme de l'Horloge qui utilisent explicitement la fonction TIME$ du PET. Ces routines devront être adaptées si l'on utilise un autre système avec horloge. La première routine lit l'horloge et renvoie les heures, minutes et secondes en HR$, MN$ et SC$ La seconde règle TIME$ à partir de l'heure « littérale » TM$. Le troisième attend que TIME$ soit différent de TT$. TT$ est la variable caractères dans laquelle on lit TIME$ chaque fois qu'on appelle la première des routines ci-dessus.

* Sur TRS-80, HR$ est obtenu en faisant MID$(TT$,10,2) et MN$ en faisant MID$(TT$,13,2).
** Sur TRS-80, il n'est pas possible d'affecter une valeur à TIME$ de cette façon.

Figure 3.15 : L'Horloge — Les routines TIME$

le format « chaîne de caractères » utilisé partout dans le programme, même s'il fallait l'adapter à un format d'horloge complètement différent. La conversion nécessaire pourrait être réalisée en modifiant les routines de la figure 3.15.

La figure 3.16 montre la routine d'initialisation du programme « Horloge ». Si l'on ne désire pas disposer d'un réglage de vitesse incorporé, la première constante de l'instruction DATA de la ligne 1150 doit être remplacée par − 1.

Ceci termine notre examen du programme « Horloge ». On pourrait lui apporter de nombreux perfectionnements. En voici quelques-uns :

- Incorporer la date. Dès qu'il a connaissance de celle-ci, le programme peut assurer sa mise à jour indéfiniment.

```
# Routine d'initialisation

init  OE = − 1                          # pas de contrôle d'événement antérieur
      AL = − 1                          # pas d'alarme
      AL$ = " inv ALARM norm "          # " ALARM " en vidéo inverse
      READ TL, TC
      DATA lignheure, colonnheure
      READ AR , AC
      DATA lignalarme, colonnalarme
      READ FT, IN                       # réglage de vitesse incorporé
      DATA intervalle, increment        # FT = intervalle. IN = 1 pour F, − 1 pour S.
      GOSUB parareglage
      RETURN

1110   OE=−1:AL=−1
1120*  AL$=CHR$(18)+"ALARME"+CHR$(146)
1130** READ TL,TC:DATA 22,28
1140   READ AR,AC:DATA 11,17
1150   READ FT,IN:DATA 2900,−1:GOSUB 510
1160   RETURN
```

Voici la routine d'initialisation de l'Horloge. Le message d'alarme est noté ainsi que les positions d'affichage pour l'heure et pour l'alarme. Puis le réglage de vitesse incorporé est initialisé. L'idée sous-jacente est qu'après avoir expérimenté les commandes F et S, l'utilisateur trouvera le réglage correct et l'incorporera au programme. C'est ainsi que l'ordinateur de l'auteur, un Pet, a besoin d'être ralenti d'une seconde toutes les 2 900 secondes environ (ligne 1150).

* Sur TRS-80, les caractères " inv " et " norm " n'existent pas.
** Sur TRS-80, la position similaire pour l'affichage de l'heure, serait obtenue avec un TL de 14 et un TC de 52.

Figure 3.16 : L'Horloge — Initialisation

- Utiliser le dispositif de contrôle d'événements pour mettre en œuvre une alarme qui consiste en une séquence d'actions planifiées, de durée, soit fixée à l'avance, soit indéterminée, à charge pour l'utilisateur de l'arrêter (par exemple, clignotement à la fréquence d'un par seconde).
- Mettre en œuvre des alarmes multiples. Permettre d'associer un message à chacune des alarmes (par exemple, " IL EST TEMPS DE REGARDER ATTENTIVEMENT STAR TREK"). Faire les modifications correspondantes dans la définition et la structure des commandes Q et R.
- Avoir la possibilité de régler l'alarme plus de 24 h à l'avance.
- Pouvoir préciser la période à la fin de laquelle l'alarme se déclenche à nouveau. (Elle est actuellement réglée pour se répéter 24 h plus tard, à moins d'avoir été annulée par la commande Q.)
- Ajouter d'autres formats pour la saisie de l'heure dans le programme de la figure 3.9.
- Avoir la possibilité d'afficher simultanément différentes heures, chacune ayant un message d'identification comme par exemple, " PARIS ", " HEURE DE MAMAN " ou " GMT ".
- Modifier le programme « Horloge » pour utiliser la variable TI du Pet de préférence à TIME$.

La Mémoire des Cartes

Nous allons maintenant examiner un autre jeu dans lequel le facteur temps est important. La Mémoire des Cartes est un jeu simple qui utilise la variable TIME$ du Pet pour mesurer des intervalles de temps. Voici comment il fonctionne :

Le programme va afficher une série de noms de cartes à jouer (par exemple, DAME DE TREFLE, 4 DE CARREAU). Il commence par demander combien de cartes vous désirez voir et l'intervalle de temps entre chacune d'elles. Par exemple,

```
NOMBRE DE CARTES ? 5
INTERVALLE (SEC) ? 3
```

fera en sorte que le programme affiche une suite de 5 cartes différentes, choisies au hasard. Chaque carte restera affichée pendant (environ) 3 secondes, puis l'écran sera effacé et la carte suivante sera affichée.

Après l'affichage des 5 cartes, le programme vous demande le nom de celles que vous avez vues et leur ordre d'apparition. Chaque fois

que le programme demande « carte ? », vous devez entrer au clavier un caractère unique indiquant la couleur de la carte (T, K, C, P) puis un espace et ensuite une lettre ou un nombre pour sa valeur (A, R, D, V, 10, 9, 8, 7, 6, 5, 4, 3, 2). Quand vous appuyez sur RETURN après avoir entré une carte selon le format ci-dessus, le programme affiche la carte exacte, et vous demande quelle est la carte suivante. A tout moment, vous pouvez, en entrant la lettre « Q » à la place du code de la carte, faire savoir au programme que vous abandonnez. Il affichera alors les cartes restantes.

A la fin de cette suite de questions, le programme dit combien de cartes vous avez nommées sans faire de faute et affiche des statistiques sur vos résultats passés. Par exemple, il pourrait dire

VOUS N'EN AVEZ TROUVE AUCUNE !

MEILLEUR RESULTAT POUR CETTE SERIE : 2 SUR 5
LONGUEUR RECORD : 3 (INTERVALLE : 2 SEC)

Ceci signifie que depuis que vous demandez au programme des suites de 5 cartes séparées par des intervalles de 3 s, votre meilleur résultat a été de 2 cartes correctes sur 5. Par ailleurs, dans le passé vous avez demandé des suites de 3 cartes à 2 s d'intervalle, et vous vous êtes souvenu de ces 3 cartes au moins une fois.

Arrivé en ce point, le programme attend que vous entriez un caractère. Si vous tapez « R », le programme affichera la suite complète pour que vous puissiez la revoir. Si vous tapez « N », le programme vous autorisera à entrer de nouvelles valeurs pour indiquer le nombre de cartes et l'intervalle. Si vous appuyez sur la barre d'espacement, le programme affichera une nouvelle suite de cartes.

Le Programme « Mémoire des Cartes »

La figure 3.17 montre la routine principale. La structure de cette routine reflète celle du jeu telle qu'elle vient d'être décrite.

L'un des sous-programmes appelés par la routine principale est montré sur la figure 3.18. Ce sous-programme simule le battage d'un paquet de cartes, mais seules les NC premières cartes sont réellement placées dans le tableau DK suivant l'ordre résultant du battage.

Le battage est effectué sans tricher. D'abord les 52 cartes sont placées dans l'ordre « standard », dans le tableau DD. Puis un nombre entier, ZZ, compris entre 1 et 52, est tiré au hasard. La carte DD(ZZ) devient le premier élément de la table DK. Tous les éléments restant de DD dont les indices sont supérieurs à ZZ, sont décalés d'une place

vers le début de la table, de sorte que les 51 premiers éléments de DD contiennent maintenant les 51 cartes restantes.

Le processus est réitéré, ZZ étant pris entre 1 et 51 pour choisir une carte DD(ZZ) qui devienne DK(2). Les éléments d'indices supérieurs à ZZ sont décalés, le tableau DD se réduit à 50 cartes et ainsi de suite.

```
# La mémoire des cartes

    GOSUB init                            # initialiser
    repeat {                              # boucle pour nouveaux types parties
        GOSUB effacecran
        GOSUB paramètres                  # nombre de cartes et durée de
                                          présentation
        repeat {                          # boucle pour un type de partie
            GOSUB effacecran
            GOSUB débutpartie             # initialisation cette partie
            GOSUB battre                  # battre le paquet
            GOSUB montrercartes           # montrer cartes une à une
            GOSUB interroger              # demander répéter
            GOSUB statis                  # dire combien de correctes
            repeat {
                GOSUB suivant             # clore la partie et continuer
                IF NX$ = " R " THEN
                    GOSUB montrersuite
                } until (NX$ <> " R")
            } until (NX$ = " N " or " E ")  # N = nouveaux paramètres
        } until (NX$ = " E")              # E = retour sous BASIC
    END


100 GOSUB 960
110 GOSUB 860:GOSUB 920
120 GOSUB 860:GOSUB 910:GOSUB 180:GOSUB 240
130 GOSUB 390:GOSUB 660
140 GOSUB 830:IF NX$="R" THEN GOSUB 270:GOTO 140
150 IF NX$="N" THEN 110
160 IF NX$="E" THEN END
170 GOTO 120
```

Le joueur choisit le nombre de cartes de la suite (de 1 à 52) et la durée de présentation DC. Cette durée est un nombre entier supérieur ou égal à zéro. Des suites de cartes sont alors montrées au joueur. Chaque carte disparaît à la fin de la durée de présentation (intervalle) DC. Enfin le joueur est prié de bien vouloir répéter la suite présentée. Après chaque carte indiquée par le joueur, le programme affiche sur l'écran la carte qui avait été effectivement présentée.

Figure 3.17 : La Mémoire des Cartes

Les sous-programmes qui montrent au joueur la série de cartes, se trouvent en figure 3.19. Ces routines sont simples et n'ont pas besoin d'autre commentaire.

La transformation du code numérique de la carte (nombre compris entre 1 et 52) en son format d'affichage (chaîne de caractères comme par exemple, « PIQUE 6 ») est effectuée par les routines de la figure 3.20. Les codes 1 à 13 ont été attribués aux trèfles, 14 à 26 aux carreaux, 27 à 39 aux cœurs et 40 à 52 aux piques. A l'intérieur de chaque groupe, la valeur de la carte est attribuée dans l'ordre As, deux, trois, quatre, cinq, six, sept, huit, neuf, dix, valet, dame, roi.

La figure 3.21 nous montre la routine qui demande au joueur de répéter le nom des cartes qui viennent d'être présentées. La

```
# Battre le paquet
battre  FOR XX = 1 TO 52                        # d'abord mettre cartes dans
                                                  l'ordre
            DD(XX) = XX,
            NEXT XX
        FOR XX = 1 TO NC                        # puis les tirer au hasard
            YY = 53 - XX                        # nombre de cartes restantes
            ZZ = INT(RND(1) * YY) + 1           # 1 < = ZZ < = YY
            DK(XX) = DD(ZZ)                     # carte suivante de la suite
            IF XX < > 52 AND ZZ < YY THEN
                FOR WW = ZZ TO YY - 1           # combler le vide laissé par
                    DD(WW) = DD(WW + 1)         # carte choisie
                    NEXT WW
            NEXT XX
        RETURN

180  FOR XX=1 TO 52:DD(XX)=XX:NEXT XX
190  FOR XX=1 TO NC:YY=53-XX
200* ZZ=INT(RND(1)*YY)+1:DK(XX)=DD(ZZ)
210  IF XX=52 OR YY<=ZZ THEN 230
220  FOR WW=ZZ TO YY-1:DD(WW)=DD(WW+1):NEXT WW
230  NEXT XX:RETURN
```

Cette routine affecte aux NC premiers éléments du tableau DK, des nombres au hasard différents tirés entre 1 et 52.

* Sur TRS-80 utiliser RND(0) au lieu de RND(1).

Figure 3.18 : La Mémoire des Cartes — Le battage des cartes

```
# Afficher les NC premières cartes (intervalle DC)

montrercartes   FOR I = 1 TO NC
                      CD = DK(I) : GOSUB affichercarte
                      DL = DC : GOSUB attente
                      GOSUB effaçecran
                      NEXT I
                RETURN


# Afficher l'ensemble des cartes

montrersuite    FOR I = 1 TO NC
                      CD = DK(I) : GOSUB affichercarte
                      NEXT I
                RETURN


# Afficher une seule carte CD

affichercarte   GOSUB decodage               # convertir CD en chaîne
                PRINT CD$
                RETURN
```

```
240  FOR I=1 TO NC
250  CD=DK(I):GOSUB 290:DL=DC:GOSUB 870:GOSUB 860:NEXT I
260  RETURN
270  GOSUB 860:FOR I=1 TO NC:CD=DK(I):GOSUB 290
280  NEXT I:RETURN
290  GOSUB 310:PRINT CD$
300  RETURN
```

Les deux premières routines présentées ici servent à afficher la suite complète des cartes de la partie en cours. La première affiche les cartes une par une, en effaçant l'écran à chaque expiration de la durée spécifiée DC. La seconde les affiche toutes ensemble pour que le joueur puisse les revoir. La troisième routine, appelée par les deux premières, transforme un nombre, compris entre 1 et 52, en un nom de carte (par exemple PIQUE 6) et affiche ce dernier.

Figure 3.19 : La Mémoire des Cartes — Affichage des cartes

réponse du joueur est saisie et sa syntaxe contrôlée. Si le joueur a entré un " Q ", toutes les cartes sont affichées. Sinon la réponse proposée est comparée à la solution correcte et le programme prend note de l'exactitude ou de l'inexactitude de la réponse. Ensuite le programme affiche la solution, efface l'écran et demande quelle est la carte suivante. La routine de la figure 3.21 appelle celles des figures 3.22 et 3.23 pour réaliser la saisie des cartes proposées par le joueur et le contrôle de syntaxe. Ces programmes s'expliquent pratiquement par eux-mêmes.

```
# Routines de Codage/Décodage

decodage  FV = mod(CD, 13)                        # valeur
          SU = (CD − FV)/13 + 1                   # couleur
          IF FV = 0 THEN
               { FV = 13 : SU = SU − 1}
          FV$ = FV$(FV) : SU$ = SU$(SU)           # mettre sous forme de chaînes
          CD$ = SU$ + " " + FV$                   # en faire un nom de carte
          RETURN

codage    IF NOT (1 < = SU < = 4 AND 1 < = FV < = 13) THEN
               CD = − 1                           # valeur ou couleur erronée
          else
               CD = 13 * (SU − 1) + FV
          RETURN

310  FV=CD−INT(CD/13)*13
320  SU=(CD−FV)/13+1
330  IF FV=0 THEN FV=13:SU=SU−1
340  FV$=FV$(FV):SU$=SU$(SU)
350  CD$=SU$+" "+FV$:RETURN
360  IF SU<1 OR SU>4 OR FV<1 OR FV>13 THEN CD=−1:RETURN
370  CD=13*(SU−1)+FV
380  RETURN
```

La première des deux routines présentées ici, transforme un nombre entier en une couleur (trèfle, carreau, cœur, pique) et une valeur (As, Roi, Dame, Valet, 10 . . . 2). Les couleurs sont codées dans SU de la façon suivante 1 = trèfles, 2 = carreaux, 3 = cœurs, 4 = piques). Les valeurs sont codées dans FV comme suit : 1 = As, 2 à 10 sans changement, 11 = Valet, 12 = Dame, 13 = Roi.

La seconde routine transforme de façon inverse les variables SU et FV en un nombre CD compris entre 1 et 52.

Figure 3.20 : La Mémoire des Cartes — Routines de codage/décodage

```
# Demander de repéter suite montrée

interroger  PRINT " REPETEZ CARTES DANS L'ORDRE "
            FOR I = 1 TO NC
                 repeat {                                # saisie réponse joueur
                      GOSUB nomcarte                     # réponse joueur dans
                                                         CD$
                      GOSUB numerocarte                  # passer de CD$ à CD
                      } until (CD < > - 1                # - 1 = erreur saisie
                 IF CD = - 2 THEN                        # joueur abandonne (Q)
                      { GOSUB montrersuite : RETURN }    # montrer toutes les
                                                         cartes
                 else IF CD = DK(I) THEN
                      GOSUB exact                        # bon coup
                 else {
                      GOSUB faux                         # mauvais coup
                      CD = DK(I)                         # afficher la bonne
                      }
                 GOSUB affichercarte                     # montrer la carte
                 DL = SC : GOSUB attente                 # (brève)
                 GOSUB effacecran
                 NEXT I
            RETURN

right       RG = RG + 1 : RETURN

wrong       RETURN
```

```
390 PRINT "REPETEZ CARTES DANS L'ORDRE"
400 FOR I=1 TO NC
410 GOSUB 490:GOSUB 500:IF CD=-1 THEN 410
420 IF CD=-2 THEN GOSUB 270:RETURN
430 IF CD=DK(I) THEN GOSUB 470:GOTO 450
440 GOSUB 480:CD=DK(I)
450 GOSUB 290:DL=SC:GOSUB 870:GOSUB 860:NEXT I
460 RETURN
470 RG=RG+1:RETURN
480 RETURN
```

Cette routine saisit la carte répétée par le joueur. Après chaque annonce (qu'elle soit bonne ou mauvaise), la carte correcte est affichée. Si le joueur tape "Q", toutes les cartes sont affichées et le processus s'arrête.

Figure 3.21 : La Mémoire des Cartes — Saisie de la réponse du joueur

```
# Demander le nom de la carte
# Format prévu : couleur (1 caractère), espace, valeur (1 ou 2 caractères)
#                la couleur est T, K, C, P
#                la valeur est A, 2, 3, 4, 5, 6, 7, 8, 9, 10, V, D, R

nomcarte    INPUT " CARTE "; CD$
            RETURN
```

```
490* INPUT "CARTE";CD$:RETURN
```

Cette routine demande il saisit la carte mémorisée par le joueur. Le contrôle de validité est indiqué en figure 3.23.

* La version Apple de cette ligne n'en diffère que par la valeur de la constante littérale.

Figure 3.22 : La Mémoire des Cartes — Entrée de la réponse

```
500 IF LEN(CD$)>=3 THEN 530
510 IF CD$="Q" THEN CD=-2:RETURN
520 CD=-1:RETURN
530 L$=LEFT$(CD$,1):R$=MID$(CD$,3):VR=VAL(R$)
540 IF L$="T" THEN SU=1:GOTO 590
500 IF L$="K" THEN SU=2:GOTO 590
560 IF L$ ="C" THEN SU=3:GOTO 590
570 IF L$ ="P" THEN SU=4:GOTO 590
580 SU=-I
590 IF VR>0 THEN FV=VR:GOTO 650
600 IF R$="A" THEN FV=1:GOTO 650
610 IF R$="R" THEN FV=13:GOTO 650
620 IF R$="D" THEN FV=12:GOTO 650
630 IF R$="V"THEN FV=11:GOTO 650
640 FV=-1
650 GOSUB 360:RETURN
```

Figure 3.23a : La Mémoire des Cartes — Contrôle de la réponse *(BASIC réel)*

```
# Transformer carte indiquée CD$ en nombre CD


numerocarte  IF LEN(CD$) < 3 THEN        # réponses ont 3 ou 4 car.
                IF CD$ = " Q " THEN
                    CD = - 2              # joueur abandonne
                else
                    CD = - 1              # réponse non valide

             else {
                L$ = LEFT$(CD$, 1)        # chaîne « couleur »
                R$ = MID$(CD$, 3)         # chaîne « valeur »
                VR = VAL(R$)
                IF case
                    L$ = " T " THEN SU = 1
                    L$ = " K " THEN SU = 2
                    L$ = " C " THEN SU = 3
                    L$ = " P " THEN SU = 4
                else
                    SU = - 1              # couleur non valide
                IF VR > 0 THEN            # R$ est numérique
                    FV = VR
                else IF case
                    R$ = " A " THEN FV = 1
                    R$ = " R " THEN FV = 13
                    R$ = " D " THEN FV = 12
                    R$ = " V " THEN FV = 11
                else
                    FV = - 1              # valeur non valide
                GOSUB codage              # SU, FV sont combinés pour donner
                    }                       CD
                RETURN
```

La routine décompose CD$ en SU et FV et appelle ensuite l'une des routines présentées à la figure 320 pour calculer CD.

Figure 3.23b : La Mémoire des Cartes — Contrôle de la réponse *(BASIC Libre)*

La figure 3.24 montre la routine qui donne le résultat (c'est-à-dire le degré d'exactitude de vos réponses). Comme pour d'autres programmes du même genre dans ce livre, cette routine a été conçue pour varier et augmenter l'intérêt du jeu. La figure 3.25 montre le sous-programme appelé par le programme de la figure 3.24 pour l'affichage des meilleurs résultats des parties passées.

```
# Afficher résultats partie et comparer avec les « meilleurs »

statis  PRINT                                # interligne
        IF RG = NC THEN                      # trouvé en totalité
            IF NC = 1 THEN
                PRINT " VOUS L'AVEZ TROUVÉE! "
            else IF NC = 2 THEN
                PRINT "VOUS LES AVEZ TROUVÉES TOUTES LES DEUX!"
            else
                PRINT "VOUS LES AVEZ TROUVEES TOUTES LES"; NC
        else IF RG = 0 THEN                  # joueur a tout raté
            IF NC = 1 THEN
                PRINT " VOUS NE L'AVEZ PAS TROUVEE! "
            else
                PRINT " VOUS N'EN AVEZ TROUVE AUCUNE ! "
        else
            PRINT " VOUS EN AVEZ TROUVE "; RG; " SUR "; NC
        PRINT                                # interligne
        IF RG = NC AND NC > BC THEN {
            PRINT "NOUVEAU RECORD DE LONGUEUR "
            BC = NC : BD = DC                # record « absolu » (longueur et intervalle)
            }
        IF RG > BR THEN
            BR = RG                          # record pour série en cours
        PRINT
        GOSUB records                        # rappeler records en cours
        RETURN
```

Cette routine donne les résultats de la partie. Puis les meilleurs scores à cet instant sont rappelés (plus longue suite mémorisée et nombre de cartes le plus grand mémorisé dans la série en cours).

Figure 3.24 : La Mémoire des Cartes — Affichage du score

```
660   PRINT:IF RG <> NC THEN 700
670   IF NC=1 THEN PRINT"VOUS L'AVEZ TROUVEE!":GOTO 740
680   IF NC=2 THEN PRINT"VOUS LES AVEZ TROUVÉES TOUTES LES DEUX!":GOTO 740
690*  PRINT"VOUS LES AVEZ TROUVÉES TOUTES LES";NC:GOTO 740
700   IF RG <> 0 THEN 730
710   IF NC=1 THEN PRINT"VOUS NE L'AVEZ PAS TROUVÉE!":GOTO 740
720   PRINT"VOUS N'EN N'AVEZ TROUVÉ AUCUNE!":GOTO 740
730*  PRINT"VOUS EN AVEZ TROUVÉ";RG;"SUR";NC
740   PRINT
750   IF RG<>NC OR NC<=BC THEN 780
760   PRINT"NOUVEAU RECORD DE LONGUEUR!"
770   BC=NC:BD=DC
780   IF RG>BR THEN BR=RG
790   PRINT:GOSUB 800:RETURN
```

---

* Les versions Apple de ces lignes n'en diffèrent que par les espaces dans les constantes littérales.

Figure 3.24a : La Mémoire des Cartes — L'affichage du score *(BASIC réel)*

```
# Rappeler records en cours

records  PRINT " MEILLEUR RESULTAT POUR CETTE SERIE:";BR;" SUR "; NC
         IF BC > 0 THEN
              PRINT" LONGUEUR RECORD : "; BC ; " (INTERVALLE : " ; BD ; " SEC)"
         RETURN

800*  PRINT"MEILLEUR RÉSULTAT POUR CETTE SERIE:";BR;"SUR";NC
810*  IF BC>0 THEN PRINT:PRINT"LONGUEUR RECORD:";BC;"(INTERVALLE:";
      BD;"SEC)"
820   RETURN
```

Cette routine affiche les meilleurs résultats obtenus jusqu'à présent.

---

* Les versions Apple de ces lignes n'en diffèrent que par les espaces dans les constantes littérales.

Figure 3.25 : La Mémoire des Cartes — Rappel des meilleurs résultats

La figure 3.26 présente plusieurs routines que nous avons utilisées dans d'autres programmes. Les deux premières mettent en œuvre la routine d'attente qui est appelée après l'affichage des résultats du jeu. La troisième sert à effacer l'écran.

La figure 3.27 montre une routine qui attend que le temps donné par TIME$, ait changé de DL secondes. Une autre version de cette routine est donnée à la figure 3.34. Notons en passant que la routine de la figure 3.27 est la seule dans le jeu « La Mémoire des Cartes » où TIME$ soit utilisé. Lorsque l'ordinateur n'a pas d'horloge, la fonction de cette routine peut être réalisée par la boucle suivante :

```
FOR XX = 1 TO N * DL : NEXT XX
```

La valeur de N peut être déterminée par approximations successives.

```
# Routines diverses (vieilles connaissances)

suivant      GOSUB uncar                 # terminer partie, dire quoi faire
             NX$ = X$
             RETURN

uncar        repeat {                    # entrée caractère unique
                 X$ = CHR$(RND(1))       # faire fonctionner RND pendant l'attente
                 GET X$
                 } until (X$ <> "")
             RETURN

effacecran   PRINT "clr";                # effacer l'écran
             RETURN


830 GOSUB 840:NX$=X$:RETURN
840 X$=CHR$(RND(1)):GET X$:IF X$="" THEN 840
850 RETURN
860 PRINT CHR$(147):RETURN
```

Voici les routines qui permettent l'entrée d'une commande (1 seul caractère) à la fin de la partie ainsi que l'effacement de l'écran. Elles suivent ici la syntaxe du Pet. Les versions Apple et TRS-80 sont identiques aux routines présentées aux figures 2.11 et 2.12.

Figure 3.26 : La Mémoire des Cartes — Routines diverses *(vieilles connaissances)*

Finalement, la figure 3.28 montre trois routines d'initialisation pour « La Mémoire des Cartes ». La première routine (ligne 960) est appelée une seule fois au lancement du programme. La seconde routine (ligne 920) est appelée chaque fois qu'il faut entrer de nouveaux paramètres, c'est-à-dire au début et chaque fois que le joueur tape " N ". La troisième routine (ligne 910) est appelée à chaque affichage d'une nouvelle série de cartes.

```
# Attendre qu'il se soit écoulé DL secondes — Version Pet

attente     DX$ = TIME$
            GOSUB convertsec : DZ = DX
            repeat {
                DX$ = TIME$
                GOSUB convertsec
                } until (DX - DZ > = DL)
            RETURN

convertsec  DX = 3600 * VAL(LEFT$(DX$, 2)) +
                   60 * VAL(MID$(DX$, 3, 2)) +
                        VAL(RIGHT$(DX$, 2))
            RETURN
```

```
870  DX$=TIME$:GOSUB 890:DZ=DX
880  DX$=TIME$:GOSUB 890:IF DX-DZ<DL THEN 880
885  RETURN
890  DX=3600*VAL(LEFT$(DX$,2))+60*VAL(MID$(DX$,3,2))+VAL(RIGHT$(DX$,2))
900  RETURN
```

Cette routine note tout de suite l'heure, puis attend que celle-ci se soit modifiée de DL secondes avant de retourner.

Les versions Apple et TRS-80 doivent utiliser une « boucle d'attente ». Les instructions BASIC (réel) pour ces deux systèmes sont :

```
870  FOR DX=1 TO DL*SS:NEXT DX:RETURN
```

La valeur de la variable SS, déterminée empiriquement, est fixée au niveau de la routine d'initialisation (fig. 3.28).

Figure 3.27 : La Mémoire des Cartes — Réalisation du délai d'attente

Comme pour les autres jeux présentés dans ce livre, on peut améliorer la Mémoire des Cartes de nombreuses façons. Voici quelques-unes des possibilités :

- Afficher l'image de chaque carte à la place de son nom. Sur le Pet, la modification correspondante pourrait être faite en remplaçant à la figure 3.19 l'instruction

  GOSUB 310 : PRINT CD$

  à la ligne 290 par

  PRINT CP$(CD),

  où CP$ est un tableau de chaînes de caractères représentant les cartes (utilisant les caractères graphiques et de déplacement

```
# Routines d'initialisation

debutpartie   RG = 0
              RETURN

parametres    repeat
                   INPUT " NOMBRE DE CARTES "; NC
                   until (1 < = NC < = 52)
              INPUT " INTERVALLE (SEC)"; DC      # Durée pour mémoriser
              SC = 1                             # Durée pour montrer bonne
                                                 réponse
              BR = 0                             # Initialisation « record pour
                                                 cette série »
              RETURN

init          DIM SU$(4), FV$(13), DK(52), DD(52)
              FOR I = 1 TO 4
                   READ SU$(I)
                   DATA "TREFLE", " CARREAU ", "COEUR", "PIQUE"
                   NEXT I
              FOR I = 1 TO 13
                   READ FV$(I)
                   DATA " AS ", " 2 ", " 3 ", " 4 ", " 5 ", " 6 ", " 7 ",
                        " 8 ", " 9 ", " 10 ", " VALET ", " DAME ", " ROI "
                   NEXT I
              BC = 0 : BD = 0                    # initialiser « record absolu »
              RETURN
```

Figure 3.28 : La Mémoire des Cartes — Initialisation

du curseur du Pet). CP$ devrait être initialisé dans la routine correspondante à la ligne 960 (fig. 3.28). Naturellement, vous aurez à tenir compte du fait que les commandes Q et R étaient prévues pour l'affichage des noms de cartes et non pour celui de leurs images. Sur Apple, vous pourriez utiliser les possibilités graphiques en couleurs. Ceci demanderait une réorganisation importante et un nouveau codage.

- Améliorer les routines d'entrée des données. Par exemple, vous pourriez afficher un nombre indiquant le rang de la carte : première, seconde ; dont le nom doit être maintenant indiqué par le joueur. Vous pourriez prévoir un code qui permette au joueur de dire « J'ai oublié celle-là. Montrez-la moi, puis laissez-moi tenter ma chance avec la suivante ».
- Améliorer les possibilités du programme en matière de statistiques. Garder la trace des meilleurs résultats pour chaque intervalle. Garder les cinq derniers résultats. Calculer les moyennes.

```
 910  RG=0:RETURN
 920* INPUT "NOMBRE DE CARTES";NC:IF NC<1 OR NC>52 THEN 920
 930* INPUT "INTERVALLE (SEC)";DC
 940  SC=1
 950  BR=0:RETURN
 960  DIM SU$(4),FV$(13),DK(52),DD(52)
 970  FOR I=1 TO 4:READ SU$(I)
 980  DATA "TREFLE","CARREAU","COEUR","PIQUE"
 990  NEXT I
1000  FOR I=1 TO 13:READ FV$(I)
1010  DATA "AS","2","3","4","5","6","7","8","9","10","VALET","DAME","ROI"
1020  NEXT I
1030  BC=0:BD=0:RETURN
```

* Les versions Apple ne diffèrent que par les valeurs des constantes littérales. Les versions Apple et TRS-80 comportent une ligne supplémentaire pour fixer la valeur de SS (voir figure 3.27). La version Apple est :

```
965  READ SS:DATA 700
```

La version TRS-80 est :

```
965  READ SS:DATA 200
```

Figure 3.28a : La Mémoire des Cartes — Initialisation *(BASIC réel)*

- Synchroniser l'affichage de la première carte. Tel que le programme est écrit actuellement, les instructions

  DL = DC : GOSUB 870

  de la ligne 250 (fig. 3.19) génèrent un délai qui peut parfois être trop court de près d'une seconde. Pour améliorer ce point, l'instruction

  FOR I = 1 TO NC

  de la ligne 240 peut être précédée par un appel à une routine qui attend le prochain battement de l'horloge.

Le Flicker* à Dix Touches

Le Flicker à Dix Touches est un de ces jeux frustrants où le programme vous pose un problème et vous fixe un délai de réponse. Si vous ne répondez pas dans le temps imparti, le programme vous fait une remarque farfelue et vous laisse tomber. Voici comment cela fonctionne :

La figure 3.29 montre une partie du clavier du Pet. Les chiffres de 1 à 9 sont disposés en carré, ce qui fournit au joueur un moyen commode de désigner dans un carré de 9 chiffres affiché sur l'écran, un emplacement particulier. Le programme commence par vous demander :

INTERVALLE ?

Vous répondez par un nombre entier qui indique le nombre de secondes (zéro compris) dont vous désirez disposer pour répondre. (Si vous pouvez gagner avec un délai de zéro seconde, vos réflexes sont à proprement parler, surhumains.) Une fois que le délai a été entré, le programme affiche sur l'écran une configuration de 9 chiffres. Ou bien les 9 chiffres sont identiques et vous devez appuyer sur la touche « 0 » avant la fin du délai, ou bien l'un des chiffres est différent des huit autres, auquel cas vous devez « désigner » le chiffre « étranger » en enfonçant la touche dont la position dans le carré du clavier numérique, correspond à celle que le chiffre « étranger » occupe sur l'écran. (Voir figure 3.30.)

Si votre réponse arrive avant la fin du délai, le programme dira C'EST EXACT ou C'EST FAUX, selon le cas. Si vous ne répondez pas, le programme dira VOUS AURIEZ INTÉRÊT A ÊTRE PLUS RAPIDE. Si votre réponse est fausse, le programme vous donnera aussi la bonne réponse et affichera de nouveau la figure formée par les

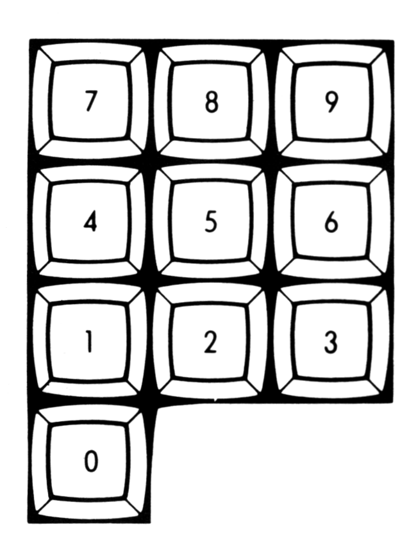
---

* de FLICK chiquenaude.

9 chiffres. Cette fois-ci vous pourrez étudier à loisir la configuration. Le programme attendra que vous entriez un seul caractère — soit en appuyant sur N pour préciser un nouveau délai de réponse, soit en enfonçant l'une des autres touches pour faire afficher la configuration suivante.

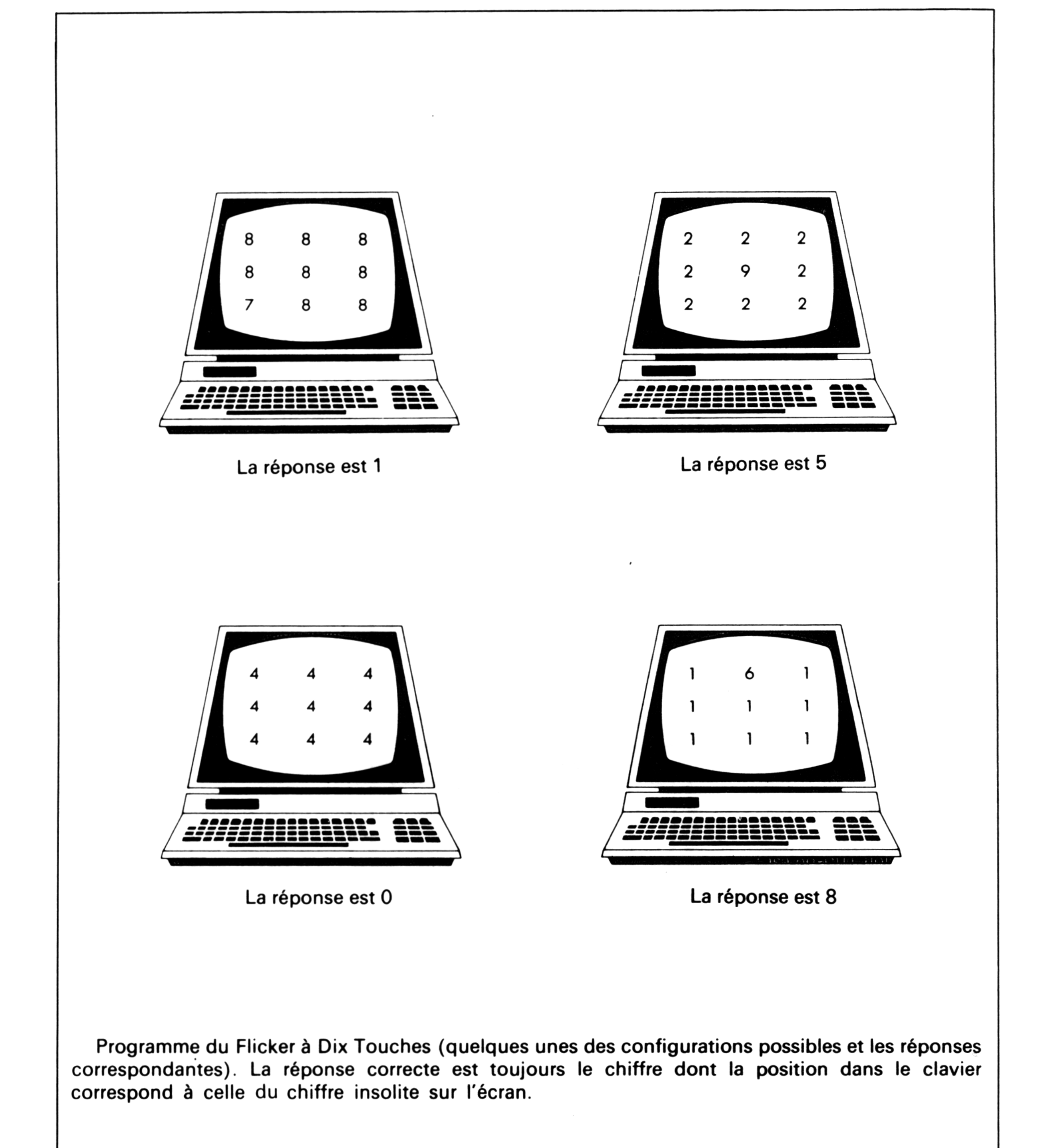
Notons en passant, qu'une réponse donnée avec un très léger retard, est interprétée par le programme comme une invitation à passer au jeu suivant. Une modification simple pourrait éliminer ce défaut, mais celui-ci donne au programme un comportement qui est dans l'esprit du jeu. (Vous pouvez modifier ce comportement en insérant GETX$ après GOSUB 300 à la ligne 160 de la figure 3.31.)

La figure 3.31 montre la partie principale du Flicker à Dix Touches. La structure a la même allure générale que celle de nombreux programmes de ce livre.



Voici une partie du clavier numérique du Pet. Le Flicker à Dix Touches met à profit la disposition en carré des touches 1 à 9 pour donner au joueur la possibilité de « montrer », dans le carré de chiffres qui est affiché un court instant sur l'écran, celui qui est différent des autres.

Figure 3.29 : Clavier Numérique du Pet



Programme du Flicker à Dix Touches (quelques unes des configurations possibles et les réponses correspondantes). La réponse correcte est toujours le chiffre dont la position dans le clavier correspond à celle du chiffre insolite sur l'écran.

Figure 3.30 : Le Flicker à Dix Touches — Exemples d'affichage

```
# Le Flicker à Dix Touches

    GOSUB init
    repeat {
        GOSUB fixdelai
        repeat {
            GOSUB configuration
            GOSUB afficher
            GOSUB entréechiffre
            IF XX = 1 THEN
                PRINT " EXACT ! "
            else IF XX = 0 THEN
                PRINT " C'EST FAUX ! etc. "; EX$;
            else                                # XX = - 1
                PRINT " IL FAUT ALLER PLUS VITE etc. "; EX$;
            DL = 2 : GOSUB attente              # attendre 2 secondes
            GOSUB afficher                      # puis réafficher configuration
            GOSUB suivant                       # attendre NX$ entré par joueur
            } until (NX$ = " N " or " E ")
        } until (NX$ = " E ")
    END
```

```
100  GOSUB 430
110  GOSUB 420
120  GOSUB 340:GOSUB 200
130  GOSUB 230:IF XX=1 THEN PRINT "EXACT!":GOTO 160
140  IF XX=-1 THEN PRINT "IL FAUT ALLER PLUS VITE!—C'ÉTAIT";EX$;:GOTO 160
150  PRINT"C'EST FAUX—LA REPONSE EST";EX$;
160  DL=2:GOSUB 300
170  GOSUB 200:GOSUB 390:IF NX$="E"THEN END
180  IF NX$="N" THEN 110
190  GOTO 120
```

Voici la routine principale du Flicker à Dix Touches. Elle ressemble aux routines principales des autres jeux présentés dans ce livre.

Figure 3.31 : Le Flicker à Dix Touches

La figure 3.32 montre la routine d'affichage de la configuration. Cette routine utilise les chaînes de caractères L$ et B$. Ces chaînes contiennent les caractères qui déplacent le curseur du Pet. Sur Apple, l'affichage de la configuration pourrait être réalisé en se servant des instructions HTAB et VTAB.

La figure 3.33 présente la routine qui saisit la réponse du joueur tout en surveillant l'heure. Cette routine utilise la variable XX pour coder les trois éventualités suivantes :

1 : Réponse correcte
2 : Mauvaise réponse
– 1 : Pas de réponse en temps voulu.

La figure 3.34 montre une routine de retardement qui remplit la même fonction que celle de la figure 3.27. Cette routine compte les battements de l'horloge ; celle de la figure 3.27 compare les temps.

```
# Afficher la configuration

afficher   PRINT " clr "
           PRINT A(7) ; L$ ; A(8) ; L$ ; A(9) ; B$ ;
                 A(4) ; L$ ; A(5) ; L$ ; A(6) ; B$ ;
                 A(1) ; L$ ; A(2) ; L$ ; A(3) ;
           RETURN
```

```
200* PRINT CHR$(147)
210  PRINT A(7);L$;A(8);L$;A(9);B$;A(4);L$;A(5);L$;A(6);B$;A(1);L$;A(2);L$;A(3);
220  RETURN
```

Cette routine affiche sur l'écran le carré de chiffres constituant la configuration.

* Sur TRS-80, utiliser CLS (voir figure 2.12).

Figure 3.32 : Le Flicker à Dix Touches — Affichage de la configuration

Saisir réponse joueur

```
entreechiffre   DL = DC
                TX$ = TIME$
                repeat {
                     GET X$
                     IF X$ <> "" THEN {
                          IF X$ = EX$ THEN
                               XX = 1
                          else
                               XX = 0
                          break
                          }
                     else if DL < = 0 then {
                          XX = - 1
                          break
                          }
                     else if TX$ <> TIME$ THEN {
                          TX$ = TIME$
                          DL = DL - 1
                          }
                     }
                PRINT "clr"
                RETURN
```

```
230   DL=DC:TX$=TIME$
240*  GET X$:IF X$<>"" THEN 280
250   IF DL<=0 THEN XX=-1:GOTO 290
260   IF TX$=TIME$ THEN 240
270   TX$=TIME$:DL=DL-1:GOTO 240
280   XX=0:IF X$=EX$ THEN XX=1
290** PRINT CHR$(147):RETURN
```

Cette routine assure la saisie d'un caractère unique entré par le joueur, à condition que la frappe ait lieu avant que l'horloge n'ait battu DC fois.

---

* Sur TRS-80, remplacer GET X$ par X$ = INKEY$. La solution serait à repenser complètement sur Apple, étant donné que l'instruction GET X$ attendrait jusqu'à ce que X$ <>"".
** Sur TRS-80, utiliser CLS (voir figure 2.12).

Figure 3.33 : Le Flicker à Dix Touches — Entrée à durée limitée

La figure 3.35 présente la routine qui crée la configuration de chiffres qui apparaîtra sur l'écran. Cette routine choisit de façon aléatoire 3 chiffres compris entre 1 et 9 :

— Le chiffre « majoritaire » qui figurera en 8 endroits.

— La position du chiffre « étranger ».

— Le chiffre « étranger ».

Si le chiffre « étranger » est le même que le chiffre « majoritaire », la réponse du joueur doit être zéro. Si le chiffre « étranger » est différent du chiffre « majoritaire » (c'est le cas le plus fréquent), le nombre indiqué par le joueur doit être le deuxième chiffre choisi au hasard.

La figure 3.36 montre les routines qui saisissent le code du prochain jeu et les deux routines d'initialisation. Dans l'initialisation se trouve incluse une définition de fonction :

```
DEF FNR9(X) = INT(RND(1) * 9) + 1
```

Cette fonction est utilisée par le programme de la figure 3.35 pour déterminer les 3 chiffres compris entre 1 et 9.

Attendre que l'horloge ait « battu » DL fois

```
attente IF DL <> 0 THEN
            FOR I = 1 TO DL
                TX$ = TIME$
                repeat {   } until (TX$ <> TIME$)
                NEXT I
            RETURN
```

```
300  IF DL=0 THEN RETURN
310  FOR I=1 TO DL:TX$=TIME$
320  IF TX$=TIME$THEN 320
330  NEXT I:RETURN
```

Cette routine attend que l'horloge ait battu DL fois. C'est une autre méthode que celle mise en œuvre à la figure 3.27.

Figure 3.34 : Le Flicker à Dix Touches — Réalisation du délai d'attente

Des améliorations peuvent être apportées au Flicker à Dix Touches. Par exemple :

- Montrer le chiffre « étranger » en vidéo inverse lorsqu'on donne la bonne réponse.
- Sonoriser le programme pour rendre celui-ci plus mal embouché et plus antipathique encore.
- Synchroniser le début de l'affichage. (Se reporter à la suggestion similaire faite pour la Mémoire des Cartes.)

```
# Créer la configuration

configuration  N1 = FNR9(1)              # tirer au hasard 3 entiers
               N2 = FNR9(1)              # compris entre 1 et 9
               N3 = FNR9(1)
               FOR I = 1 TO 9            # rendre les 9 nombres égaux à N1
                    A(I) = N1
                    NEXT I
               A(N2) = N3                # remplacer alors le N2-ième par N3
               IF N1 = N3 THEN           # prévoir le caractère unique
                    EX$ = " 0 "          # constituant bonne réponse
               else
                    EX$ = RIGHT$(STR$(N2), 1)
               RETURN
```

```
340* N1=FNR9(1):N2=FNR9(1):N3=FNR9(1)
350  FOR I=1 TO 9:A(I)=N1:NEXT I:A(N2)=N3
360  IF N1=N3 THEN EX$="0":GOTO 380
370  EX$=RIGHT$(STR$(N2),1)
380  RETURN
```

Cette routine crée la configuration que le joueur doit identifier. Un carré de 9 chiffres sera affiché (3 lignes de 3 chiffres chacune). Si les chiffres sont tous les mêmes, le joueur doit taper sur zéro. Dans le cas contraire un chiffre différera des autres et le joueur devra appuyer sur le chiffre dont la position sur le clavier numérique correspond à celle du chiffre « insolite ».

* L'argument 1 qui apparaît dans l'appel de la fonction est fictif (voir figure 3.36).

Figure 3.35 : Le Flicker à Dix Touches — Création de la configuration

```
# Saisir code partie suivante

suivant   GOSUB uncar
          NX$ = X$
          RETURN

uncar     repeat {
               X$ = STR$(RND(1))
               GET X$
               } until (X$ <> "")

# Demander valeur attente

fixdelai  INPUT " DUREE INTERVALLE "; DC
          RETURN

# Initialiser

init      DIM A(9)
          L$ = " seize caractères curseur droite "
          B$ = " un caractère curseur gauche + dix car. curseur bas "
          DEF FNR9(X) = INT(RND(1) * 9) + 1          # ne pas tenir compte de X
          RETURN
```

```
390   GOSUB 400:NX$=X$:RETURN
400*  X$=STR$(RND(1)):GET X$:IF X$="" THEN 400
410   RETURN
420   INPUT "DURÉE INTERVALLE";DC:RETURN
430   DIM A(9)
400** L$="":FOR LL=1 TO 16:L$=L$+CHR$(29):NEXTLL
450** B$=CHR$(157):FOR LL=1 TO 10:B$=B$+CHR$(17):NEXTLL
460*  DEF FNR9(X)=INT(RND(1)*9)+1
470   RETURN
```

* Sur TRS-80 mettre RND(0) au lieu de RND(1) et remplacer GET X$ par X$=INKEY$.

** Sur TRS-80 ces lignes deviennent :

```
440   L$="":FOR LL=1 TO 27:L$=L$+CHR (25):NEXT LL
450   B$=CHR$(24):FOR LL=1 TO 6:B$=B$+CHR$(26):NEXTLL
```

Figure 3.36 : Le Flicker à Dix Touches — Routines diverses

Le Chronomètre

Ce n'est pas un jeu baptisé « Chronomètre », il s'agit d'un chronomètre pour les jeux. Vous pourriez par exemple, l'utiliser pour mesurer la durée des coups dans une partie d'échecs.

Le programme (fig. 3.37) affiche un nombre dans le coin supérieur gauche de l'écran. Au départ ce nombre est zéro. Dès que vous appuyez sur la barre d'espacement, ce nombre commence à croître au rythme d'une unité par seconde, ce qui revient à dire que le programme affiche le nombre de secondes écoulé depuis que vous avez appuyé sur la barre d'espacement. Si vous appuyez à nouveau sur cette barre, le nombre cessera de croître. Mais si vous l'actionnez encore une fois, le programme reprendra le chronométrage là où il était stoppé. A condition que le chronomètre soit arrêté, vous pouvez à tout moment le remettre à zéro en enfonçant la touche R.

Il y a plusieurs façons d'améliorer ce programme :

- Afficher le temps écoulé suivant le format HH : MM : SS.
- Avoir deux chronomètres, affichés dans deux coins différents de l'écran. Appuyer sur la barre d'espacement déclenchera le premier chronomètre. Ensuite, cette barre permettra d'arrêter le chronomètre qui est en marche et simultanément de démarrer l'autre.

```
100  TX=0:GOSUB 190
110* GET X$:IF X$="" THEN 110
120  IF X$="R" THEN TX=0:GOSUB 190:GOTO 110
130  IF X$="" THEN GOSUB 150:GOTO 110
140  GOTO 110
150  TX$=TIME$
160* GET X$:IF X$<>"" THEN RETURN
170  IF TX$=TIME$ THEN 160
180  TX=TX+1:GOSUB 190:GOTO 150
190* IF TX=0 THEN PRINT CHR$(147):GOTO 200
195* PRINT CHR$(19)
200  PRINT TX:RETURN
```

* La version TRS-80 utilise X$ = INKEY$ à la place de GET X$. Le PRINT "clr" est remplacé par CLS, et le caractère « retour en haut à gauche » est obtenu par CHR$(28). Pour l'Apple il faudrait repenser complètement la solution car le GET X$ de l'Apple attendrait jusqu'à obtenir X$ < > ""

Figure 3.37a : Le Chronomètre. Listing BASIC

- Faire en sorte qu'un temps limite puisse être précisé et qu'un message ou un signal d'alarme soit affiché à l'heure prévue.
- Utiliser pour « Chronomètre » la variable TI du Pet.
- Incorporer la fonction « Chronomètre » au programme « Horloge » exposé au début de ce chapitre.

```
# Le Chronomètre

                 TX = 0                          # remettre à zéro
                 GOSUB affichercompte
                 repeat {
                      GET X$                     # attendre la commande
                      IF X$ = " R " THEN {       # R remet à zéro
                           TX = 0
                           GOSUB affichercompte
                           }
                      else IF X$ = " espace " THEN   # un espace redémarre le
                                                     comptage
                           GOSUB comptage
                      }
comptage         repeat {
                      TX$ = TIME$
                      repeat {
                           GET X$
                           IF X$ <> "" THEN      # Touche quelconque
                                                 arrête comptage
                                RETURN
                           } until (TX$ <> TIME$)
                      TX = TX + 1                # ajouter 1 pour chaque
                                                 seconde
                      GOSUB affichercompte
                      }
affichercompte   IF TX = 0 THEN                  # si redémarrage à zéro
                      PRINT " clr "              # effacer l'écran
                 else
                      PRINT " curseur en haut à gauche "
                 PRINT TX
                 RETURN
```

Voici le programme du Chronomètre.

Figure 3.37b : Le Chronomètre — Description en BASIC Libre

En résumé

Le but du présent chapitre était de donner des exemples des techniques utilisées pour contrôler l'élément temps dans les programmes de jeux. Nous avons commencé par décrire les possibilités offertes par l'horloge du Pet. Puis, nous avons examiné quatre jeux qui montrent quelques-unes des façons différentes d'utiliser l'élément temps.

Le programme « Horloge » montre les techniques à exploiter pour avoir un affichage permanent de l'heure pendant que se déroulent d'autres activités. « La Mémoire des Cartes » et « Le Flicker à Dix Touches » montrent comment se servir de l'horloge pour faire varier la vitesse d'exécution et chronométrer l'action du joueur. Le « Chronomètre » montre comment mesurer le temps à la façon d'un chronomètre à main.

CHAPITRE 4

Jeux chronologiques

Les Jours de la Semaine
L'enfant du Lundi a des traits harmonieux,
L'enfant du Mardi est très gracieux,
L'enfant du Mercredi est malheureux,
L'enfant du Jeudi vivra très vieux,
L'enfant du Vendredi est généreux,
L'enfant du Samedi est besogneux.
Et celui qui naît un Dimanche,
est beau, sage, bon et joyeux.

— *Auteur inconnu ; cité par Bray dans*
Les Traditions du Devon

Tous les programmeurs devraient être rompus aux techniques d'entrée et de sortie des dates. Les jeux de ce chapitre : L'Anniversaire et Le Calendrier, ont été conçus pour illustrer ces techniques.

L'Anniversaire

Ce jeu a pour but de vous apprendre :

— A déterminer le jour de la semaine correspondant à une date donnée. (Ce qui sert non seulement dans les jeux, mais aussi dans de nombreux calculs de gestion.)

— A mettre au point un format pratique pour l'entrée des dates.

Le programme commence par demander votre date de naissance (voir figure 4.1). Vous répondez en entrant une date sous la forme « mois, jour, année », où le mois, le jour et l'année sont des nombres séparés par des barres de fraction (ou des espaces ou des traits d'union). Si l'année est un nombre de deux chiffres, le programme y ajoutera automatiquement 1900. Dans l'exemple donné à la figure 4.1, le joueur a voulu dire 1941 en tapant " 41 ". (Si vous voulez préciser qu'il s'agit de l'an 41 après J. C., il faut frapper " 041 ".) Le programme répond à l'entrée de cette date de naissance en affichant " VOUS ETES NE LE ", suivi du jour de la semaine et de la date, comme il est montré en figure 4.1. Il poursuit alors par une remarque tout à fait gratuite en fonction du jour de la semaine (par exemple, " L'ENFANT DU LUNDI A DES TRAITS HARMONIEUX ").

Puis, le programme attend que le joueur appuie sur une touche quelconque. Cette action déclenchera l'effacement de l'écran et l'on repartira pour un tour.

Le programme de l'Anniversaire

Ce programme est présenté de la figure 4.2 à la figure 4.11. Une grande partie des routines présentées sera réutilisée dans d'autres programmes de dates.

La routine principale, présentée à la figure 4.2, est très simple et ne requiert aucun commentaire. Sa structure ressemble à celles des autres procédures figurant dans ce livre.

La routine de saisie de la date (fig. 4.3) a été écrite de manière à être très générale. Elle accepte comme argument le message qu'elle utilisera ensuite pour demander la date.

La routine qui contrôle la date est présentée à la figure 4.4. Tout d'abord, celle-ci stocke la longueur de la chaîne date, dans la variable L. Cette longueur est communiquée à une autre routine qui découpera la chaîne en trois zones. Ensuite, la routine s'assure que la date est valide (auquel cas XX est mis à 1) et retourne les valeurs du jour, du mois et de l'année (cette dernière augmentée de 1900, si nécessaire) dans les variables MO, DA et YR. Si la date n'est pas valide, la routine retourne XX = 0.

Le sous-programme de la figure 4.5 découpe la date en trois zones en fonction des séparateurs présents dans la chaîne (barres de fraction, tirets ou espaces).

```
DATE DE NAISSANCE : 8/31/41
VOUS ETES NE LE
DIMANCHE, 31 AOUT, 1941
COMME C'EST BIEN !

VOUS ETES BEAU, SAGE, BON ET JOYEUX
```

Voici un exemple de ce qu'affiche à chaque tour l'Anniversaire. Le programme attend que le joueur enfonce une touche pour effacer l'écran et passer au tour suivant.

Figure 4.1 : L'Anniversaire — Affichage

```
# L'Anniversaire
    GOSUB init
    repeat {
        GOSUB effacecran                  # commencer sur un écran
                                          propre
        DP$ = " DATE DE NAISSANCE : "     # argument de "entreedate"
        GOSUB entreedate                  # demander et saisir une date
        GOSUB jour                        # trouver jour de la semaine
        GOSUB rengainejour                # réciter poésie, etc...
        GOSUB uncar                       # attendre frappe un car.
        }

100 GOSUB610
110 GOSUB510:DP$="DATE DE NAISSANCE":GOSUB120:GOSUB300:GOSUB400:
    GOSUB520:GOTO110
```

Ceci est la routine principale de l'Anniversaire.

Figure 4.2 : L'Anniversaire

```
# Saisir date — en tirer DA, MO, YR
entreedate  repeat {
                PRINT DP$;               # appel date
                GOSUB entreechaîne       # entrée en continu jusqu'à RETURN
                GOSUB controledate       # convertir en DA, MO, YR
                } until (XX = 1)
            RETURN

120 PRINT DP$;:GOSUB540:GOSUB140:IF XX<>1 THEN120
130 RETURN
```

Voici la routine de saisie de la date. Elle utilise le libellé d'appel DP$ comme argument et donc peut servir à saisir des dates dans d'autres buts. Dans l'Anniversaire, DP$ est initialisé à " DATE DE NAISSANCE" : avant que la présente routine ne soit appelée.

Figure 4.3 : Saisie de la Date

```
# Transformer chaîne entrée XX$ en MO, DA, YR

controledate  L = LEN(XX$)
              IF L > = 5 THEN {                        # pas grave si trop court
                   GOSUB zones                         # répartir XXS entre
                                                       MO$, DA$, YR$
                   MO = INT(VAL(MO$))
                   DA = INT(VAL(DA$))
                   YR = INT(VAL(YR$))
                   IF LEN(YR$) = 2 THEN                # année à 2 chiffres YR
                                                       signifie 19YR
                        YR = YR + 1900
                   GOSUB bissex                        # si année bissextile
                   IF XX = 1 THEN
                        ML(2) = 29                     # alors Fév. a 29 jours.
                   else
                        ML(2) = 28                     # sinon 28
                   IF 1 < = MO < = 12 AND YR > = 1 THEN   # contrôle mois et
                                                            année
                        IF 1 < = DA < = ML(MO) THEN    # contrôle jour
                             IF NOT (YR = 1582 AND     # un de ceux supprimés
                                                       par Grégoire ?
                             MO = 10 AND 4 < DA < 15) THEN {
                                  XX = 1
                                  RETURN
                                  }
                   }
              XX = 0
              RETURN
```

```
140 L=LEN(XX$):IF L<5 THEN210
150 GOSUB220:MO=INT(VAL(MO$)):DA=INT(VAL(DA$)):YR=INT(VAL(YR$))
160 IF LEN(YR$)=2 THEN YR=YR+1900
170 GOSUB480:ML(2)=28:IF XX=1 THEN ML(2)=29
180 IF NOT (1<=MO AND MO<=12 AND YR>=1) THEN210
190 IF NOT (1<=DA AND DA<=ML(MO)) THEN210
200 IF NOT (YR=1582 AND MO=10 AND 4<DA AND DA<15) THEN XX=1:RETURN
210 XX=0:RETURN
```

Cette routine recherche dans la chaîne XX$, trois sous-chaînes correspondant au mois, au jour et à l'année d'une date valide à partir du 1er janvier de l'an 1 après J.-C. Cela est plus difficile qu'il n'y paraît. En effet les dates allant du 5 au 14 octobre 1582 n'existent pas.

Figure 4.4 : Contrôle de la Date

```
# Extraire de XX$, les sous-chaînes MO$, DA$, YR$ — MO$ = "" signale format faux

zones  XS = 0                                  # nombre de séparateurs trouvés
       FOR XL = 2 TO L - 1                     # impossible commencer ou finir par
                                                 séparateur
            IF MID$(XX$, XL, 1) =              # si caractère est un séparateur
            "/" or " " or "-" THEN {
                 XS = XS + 1                   # alors repérer son rang
                 IF XS = 1 or 2 THEN
                      SP(XS) = XL              # et sa position
                 else {
                      MO$ = ""
                      RETURN
                      }
                 }
            NEXT XL
       IF XS <> 2 THEN
            MO$ = ""
       else {                                  # utiliser les deux séparateurs
            MO$ = LEFT$(XX$, SP(1) - 1)        # pour trouver les trois zones
            DA$ = MID$(XX$, SP(1) + 1, SP(2) - SP(1) - 1)
            YR$ = MID$(XX$, SP(2) + 1)
            }
       RETURN
```

```
220 XS=0:FOR XL=2 TO L-1:ZZ$=MID$(XX$,XL,1)
230 IF ZZ$<>"/" AND ZZ$<>" " AND ZZ$<>"-" THEN260
240 XS=XS+1:IF XS=1 OR XS=2 THEN SP(XS)=XL:GOTO260
250 MO$="":RETURN
260 NEXT XL:IF XS<>2 THEN MO$="":GOTO290
270 MO$=LEFT$(XX$,SP(1)-1):DA$=MID$(XX$,SP(1)+1,SP(2)-SP(1)-1)
280 YR$=MID$(XX$,SP(2)+1)
290 RETURN
```

Cette routine décompose XX$ en trois sous-chaînes séparées par des barres de fraction, des tirets ou des espaces. Ces trois sous-chaînes sont placées dans les variables MO$, DA$ et YR$. Aucun contrôle n'est effectué à part celui concernant l'existence de ces trois zones dans le format donné. Dans la négative, la routine renvoie MO$="".

Figure 4.5 : Découpage de la Chaîne Date en Zones

```
# Passer de MO, DA, YR à DW (0 = Dimanche, ..., 6 = Samedi)

# Version Grégorienne — suppose un changement de calendrier en 1582

jour  YZ = YR mod 400                # 400 ans = exactement 20 871 semaines
      DW = DZ                        # 1er Janvier 2001
      while (YZ > = 101) {           # ajouter 5 jours par siècle
           YZ = −100
           DW = DW + 5
           }
      while (YZ > = 5) {             # ajouter 5 jours tous les 4 ans
           YZ = YZ − 4
           DW = DW + 5
           }
      IF YZ = 0 THEN
           DW = DW − 2               # 2 jours pour 2000 après J.-C. (par exemple)
      else
           DW = DW + (YZ − 1)        # 1 jour pour chacune des années 2002-2004
      GOSUB julien                   # JD = rang du jour dans l'année
      DW = DW + JD − 1               # ajouter un jour pour chaque jour
      IF (MO, DA, YR) < = (10, 4, 1582) THEN
           DW = DW + 3               # ajouter 3 pour les 11 jours manquant
      DW = DW (mod 7)
      RETURN
```

```
300  YZ=YR-400*INT(YR/400):DW=DZ
310  IF YZ<101 THEN330
320  YZ=YZ-100:DW=DW+5:GOTO310
330  IF YZ<5 THEN350
340  YZ=YZ-4:DW=DW+5:GOTO330
350  IF YZ=0 THEN DW=DW-2:GOTO356
352  DW=DW+YZ-1
356  GOSUB450:DW=DW+JD-1:IF YR>1582 THEN390
360  IF YR=1582 AND MO>10 THEN390
370  IF YR=1582 AND MO=10 AND DA>=15 THEN390
380  DW=DW+3
390  DW=DW-7*INT(DW/7):RETURN
```

Cette routine traite les dates égales ou postérieures au 1er janvier de l'an 1 après J.-C. et donne le jour de la semaine correspondant. Il est tenu compte du changement de calendrier décidé par le Pape Grégoire XIII, changement qui fit que le lendemain du 4 octobre 1582 fut le 15 octobre 1582.

Figure 4.6 : Trouver le Jour de la Semaine

La routine qui détermine le jour de la semaine correspondant à la date, est présentée en figure 4.6. Bien que la programmation ait été conçue pour traiter correctement les dates antérieures au changement de calendrier de 1582, cette routine donne en fait des résultats faux pour les dates inférieures à 1600. A titre d'exercice, vous pourriez essayer de trouver et corriger l'erreur qui subsiste dans cette routine et dans celle de la figure 4.9. La routine de la figure 4.6 aurait été plus compliquée, si on avait utilisé comme date de base, le 1[er] janvier 2000 au lieu du 1[er] janvier 2001. Essayez de voir pourquoi.

La routine de la figure 4.7 constitue la partie ludique du programme. Elle est extrêmement simple, car elle ne comporte que des instructions PRINT.

La routine de la figure 4.8 montre le calcul de la date « Julienne ». Celle-ci est le rang, de 1 à 365 (ou 366 lorsque l'année est bissextile), de chaque jour de l'année. La routine utilise le tableau J, qui est initialisé par la routine de la figure 4.11. Chaque élément de J correspond à un mois de l'année. La valeur de l'élément est le cumul des jours écoulés depuis le 1[er] janvier jusqu'au 1[er] du mois considéré (pour les années non bissextiles). Par exemple, J(1) = 0, car il

```
#  Dire quelque chose d'aimable à propos de la date de naissance
rengainejour PRINT : PRINT " VOUS ETES NE LE "
             PRINT DW$(DW) ; " , " ; DA ; MO$(MO) ; " , " ; YR
             PRINT "COMME C'EST BIEN ! "
             PRINT
             PRINT PM$(DS)
             RETURN

   400   PRINT:PRINT"VOUS ETES NE LE"
   410*  PRINT DW$(DW);",";STR$(DA);MO$(MO);",";YR
   420   PRINT "COMME C'EST BIEN!"
   430   PRINT
   440   PRINT PM$(DW):RETURN
```

Voici la routine qui fait une remarque à propos de la date de naissance du joueur. Le tableau des remarques pour chaque jour, PM$, est initialisé par des ordres DATA qui se trouvent dans la routine d'initialisation (fig. 4.11).

* La version Apple de cette ligne n'en diffère que par le nombre d'espaces de la variable littérale précédant YR.

Figure 4.7 : Remarque Gratuite sur la Date de Naissance

```
# Rang de jour dans l'année : convertir (MO, DA, YR) en JD

julien  JD = J(MO) + DA
        IF MO > 2 THEN {
             GOSUB bissex
             IF XX = 1 THEN
                  JD = JD + 1
             }
        RETURN

   450  JD=J(MO)+DA:IF MO<=2 THEN RETURN
   460  GOSUB480:IF XX=1 THEN JD=JD+1
   470  RETURN
```

Cette routine détermine la date « Julienne ». La variable JD prend une valeur allant de 1 à 365 (ou 366), le 1er janvier correspondant à 1, le 31 décembre à 365 ou 366.

Figure 4.8 : Numéro du Jour dans l'Année *(Date Julienne)*

```
# YR est-elle bissextile ? — Si oui XX = 1

bissex  IF 4 divise YR AND               # toutès les années divisibles par 4
        (100 ne divise pas YR OR         # sont bissextiles
        400 divise (YR) THEN             # exceptée les années centenaires
             XX = 1                      # qui ne sont pas divisibles
        else                             # par 400
             XX = 0
        RETURN

   480  XX=0:IF YR<>4*INT(YR/4)THEN RETURN
   490  IF YR=100*INT(YR/100)AND YR<> 400*INT(YR/400)THEN RETURN
   500  XX=1:RETURN
```

Cette routine indique si l'année YR est bissextile. Elle retourne XX=1 si YR est bissextile, XX=0 dans le cas contraire. Les années bissextiles sont les années divisibles par 4, à l'exception des années centenaires non divisibles par 400 (par exemple 1700, 1800, 1900 ne sont pas bissextiles). Essayer de trouver l'erreur qui se trouve dans cette routine.

Figure 4.9 : L'Année est-elle bissextile ?

n'y a pas de jours dans l'année avant janvier. J(2) = 31, J(3) = 59 (c'est-à-dire, 31 + 28) et ainsi de suite.

La routine de la figure 4.9 retourne XX = 1 ou XX = 0 suivant que l'année est bissextile ou non. La routine contient une erreur due au fait que « l'histoire a été réécrite » de façon erronée. Essayez de trouver et de corriger cette erreur. La modification s'appliquera à la routine de la figure 4.6. (Indication : ces routines traitent l'année 1500 comme l'année 1900.)

```
# Entrée chaîne en continu
entreechaîne XX$ = ""                          # commencer par chaîne nulle
             repeat {
                  GOSUB uncar                  # saisir caractère suivant
                  IF X$ <> « annuler » THEN {  # si ce n'est pas le caractère
                                                 d'effacement
                       PRINT X$;               # le reproduire sur écran
                       IF X$ <> "return" THEN  # si ce n'est pas RETURN
                            XX$ = XX$ + X$     # l'ajouter à la chaîne
                       else break              # entrée terminée par RETURN
                       }
                  else IF LEN(XX$) <> 0 THEN { # si caractère d'effacement
                       PRINT "annule la chaîne" # effacer de l'écran
                       XX$ = LEFT$(XX$, LEN(XX$)—1)  # et retirer de la
                                                       chaîne
                       }
                  }
             RETURN

540  XX$=""
550* GOSUB520:IF X$=CHR$(20) THEN580
560  PRINT X$;:IF X$=CHR$(13) THEN600
570  XX$=XX$+X$:GOTO550
580  IF LEN(XX$)=0 THEN550
590** PRINT X$;:XX$=LEFT$(XX$,LEN(XX$)-1):GOTO550
600  RETURN
```

* Sur Apple et TRS-80, le caractère d'annulation est CHR$(8).
** Sur Apple cette ligne est à remplacer par les deux lignes :

```
590  PRINT X$;" ";X$;:IF LEN(XX$)=1 THEN XX$="":GOTO550
595  XX$=LEFT$(XX$,LEN(XX$)-1):GOTO550
```

LE BASIC de l'Apple n'accepte pas une longueur nulle comme argument de la fonction LEFT$, et la « flèche arrière » de l'Apple n'efface pas le caractère figurant déjà sur l'écran.

Figure 4.10a : L'Anniversaire — Routines utilitaires

```
# Effacer l'écran

effacecran   PRINT "clr";
             RETURN

# Entrée d'un caractère unique

uncar        repeat
                 GET X$
                 until (X$ <> "")
             RETURN

   510  PRINTCHR$(147);:RETURN
   520  GET X$:IF X$=""THEN520
   530  RETURN
```

Ces routines utilitaires ressemblent aux autres routines du même genre (voir figures 2.11 et 2.12).

Figure 4.10b : L'Anniversaire — Routines utilitaires

```
# Initialisation de l'Anniversaire

   610  READ DZ:DATA 1
   620  DIM DW$(6),J(12),MO$(12),ML(12),PM$(6)
   630  FOR XX=0 TO 6:READ DW$(XX),PM$(XX):NEXT XX
   640  DATA SUNDAY,"VOUS ETES BEAU, SAGE, BON ET JOYEUX"
   650  DATA MONDAY,"L'ENFANT DU LUNDI A DES TRAITS HARMONIEUX"
   660  DATA TUESDAY,"L'ENFANT DU MARDI EST TRES GRACIEUX"
   670  DATA WEDNESDAY,"L'ENFANT DU MERCREDI EST MALHEUREUX"
   680  DATA THURSDAY,"L'ENFANT DU JEUDI VIVRA TRES VIEUX"
   690  DATA FRIDAY,"L'ENFANT DU VENDREDI EST GENEREUX"
   700  DATA SATURDAY,"L'ENFANT DU SAMEDI EST BESOGNEUX"
   710  FOR XX=1 TO 12:READ MO$(XX):NEXT XX
   720  DATA JANVIER,FEVRIER,MARS,AVRIL,MAI,JUIN,JUILLET,AOUT,SEPTEMBRE
   730  DATA OCTOBRE,NOVEMBRE DECEMBRE
   740  DT=0:FOR XX=1 TO 12:J(XX)=DT:READ ML(XX):DT=DT+ML(XX):NEXT XX
   745  IF DT<> 365 THEN STOP
   750  DATA 31,28,31,30,31,30,31,31,30,31,30,31
   760  RETURN
```

Ceci est l'initialisation de l'Anniversaire. Le BASIC et le BASIC Libre étant presque identiques, seule la version BASIC est présentée.

Figure 4.11 : L'Anniversaire — Initialisation

La figure 4.10 présente plusieurs routines utilitaires qui se trouvent sous une forme similaire ou identique dans d'autres programmes du présent ouvrage. La routine de saisie de la chaîne de caractères, présentée ici, réalise les fonctions de l'instruction BASIC

LINE INPUT XX$

Il y a peu d'ordinateurs individuels qui disposent de l'instruction LINE INPUT.

La figure 4.11 montre les initialisations qui serviront aux diverses routines de l'Anniversaire.

Le Calendrier

Ce jeu a été conçu pour vous montrer comment les routines de l'Anniversaire peuvent être utilisées dans d'autres programmes. Le jeu du Calendrier est tiré de celui de l'Anniversaire par le processus de cannibalisation que nous avons décrit au chapitre 2.

Le programme commence par demander un mois et une année sous forme de nombres comme dans l'Anniversaire. Le programme accuse réception en affichant le calendrier correspondant au mois indiqué (fig. 4.12). Le Calendrier reste affiché jusqu'à ce qu'on appuie sur une touche. L'écran est alors effacé et le programme demande qu'on lui fournisse à nouveau un mois et une année.

Le programme du Calendrier

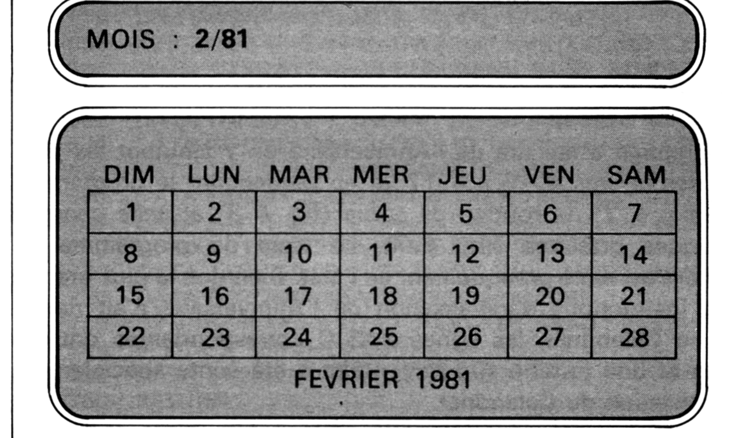
Ce programme a été tiré de l'Anniversaire en y ajoutant les programmes des figures 4.13 à 4.17, et en retranchant le programme principal (fig. 4.2), la routine de saisie (fig. 4.3) et celle chargée des remarques gratuites (fig. 4.7). Le reste du programme de l'Anniversaire est resté pratiquement en l'état. Bien que la plus grande partie des instructions d'initialisation de l'Anniversaire n'ait rien à voir avec le Calendrier, les lignes BASIC correspondantes ont été conservées et une routine supplémentaire a été écrite spécialement pour l'initialisation du Calendrier.

La routine principale du Calendrier (fig. 4.13) laisse apparaître quelques « contorsions » dues au fait que les routines de l'Anniversaire y ont été utilisées à des fins légèrement différentes de celles pour lesquelles elles avaient été prévues. C'est ainsi qu'après avoir obtenu un mois et une année sous le format " 2/81 " (février 1981),

le programme commence par le transformer en " 1/2/81 " (qui, normalement, signifie January 2, 1981). Le programme appelle alors, la routine de la figure 4.5. Celle-ci répartit la date en trois zones : MO$ = " 1 ", DA$ = " 2 ", YR$ = " 81 ". Ces trois rubriques sont réordonnées pour obtenir une chaîne dans le format adéquat : " 2 1 81 " (voulant dire February 1, 1981). La validité de cette chaîne est alors contrôlée par la routine de la figure 4.4, comme si le joueur avait, dès le début, tapé " 2/1/81 ".

Hormis ce point, les autres routines de l'Anniversaire peuvent être utilisées telles quelles. La routine de la figure 4.6 est appelée pour déterminer le jour de la semaine correspondant au 1er février 1981. Cette donnée permet de réaliser l'affichage du calendrier de façon simple, en faisant appel aux routines des figures 4.14 et 4.15. La première affiche le cadre du calendrier (noms des jours, grille, mois et année). La seconde routine inscrit les quantièmes dans ce cadre.

Le contrôle de la position du curseur permet de programmer séparément l'affichage du cadre et celui des quantièmes. Les deux routines (fig. 4.14 et 4.15) font appel à une routine qui positionne le curseur à la ligne LL et à la colonne CC. (Le « curseur » est le nom donné à la position d'impression du prochain caractère. Sa position est parfois indiquée par un carré qui clignote sur l'écran, et qui est



Voici ce qui est affiché sur écran par le jeu du Calendrier. Le joueur indique un mois et une année, et le programme affiche le calendrier qui correspond à ce mois.

Figure 4.12 : Le Calendrier — Affichage

lui-même appelé curseur.) La routine de la figure 4.15 utilise des instructions de contrôle du curseur pour amener ce dernier aux emplacements successifs de la grille où doivent être affichés les quantièmes. Les deux chaînes numériques de deux caractères cadrés à droite, sont affichées par une instruction PRINT terminée par un point-virgule. Ceci donne la certitude que rien de ce qui est déjà présent sur l'écran ne sera perturbé par l'affichage des nombres.

```
# Calendrier
    GOSUB init                                   # initialisation de " l'Anniversaire "
    GOSUB calinit                                # initialisation du "Calendrier"
        repeat {
        GOSUB effacecran                         # commencer sur écran propre

        repeat {
            PRINT "MOIS :";                      # obtenir mois et an (" MM/YY")

            GOSUB entreechaine
            XX$ = "1/" + XX$                     # remplacer par " 1/MM/YY "

            L = LEN(XX$)
            GOSUB zones
            XX$ = DA$ + " " + MO$ + " " + YR$    # en faire "MM1YY"
        GOSUB controledate                       # vérifier validité
            } until (XX = 1)                     # XX = 1 si valide
        GOSUB jour : SZ = DW                     # trouver point de départ
        NL = 1                                   # première ligne
        DL = ML(MO) - (7 - SZ)                   # contient 7 - SZ jours
        while (DL > 0) {
            DL = DL - 7 : NL = NL + 1            # chaque ligne supplémentaire en contient 7,
            }                                    # jusqu'à la derrière
        GOSUB effacecran : GOSUB cadre           # tracer NL lignes de cadres
        ND = ML(MO) : GOSUB affichquantiemes     # afficher 1 à ND
        GOSUB uncar                              # laisser sur l'écran jusqu'à touche appuyée
        }
    END
```

Ceci est la routine principale de Calendrier. Elle utilise plusieurs des routines de l'Anniversaire.

Figure 4.13 : Le Calendrier

```
100  GOSUB610:GOSUB900
105  GOSUB510
110  PRINT"MOIS:";:GOSUB540:XX$="1/"+XX$:L=LEN(XX$):GOSUB220
112  XX$=DA$+""+MO$+""+YR$:GOSUB140:IF XX<>1 THEN110
115  GOSUB300:SZ=DW:NL=1:DL=ML(MO)+SZ-7
120  IF DL>0 THEN DL=DL-7:NL=NL+1:GOTO120
125  GOSUB510:GOSUB130:ND=ML(MO):GOSUB400
128  GOSUB520:GOTO105
```

Figure 4.13a : Le Calendrier — BASIC Réel

```
# Affichage du cadre

cadre     PRINT HD$ : PRINT TL$          # noms des jours et ligne supérieure
          FOR LX = 1 TO NL
               FOR VV = 1 TO VS
                    PRINT VL$            # côtés des cases
                    NEXT VV
               PRINT BL$                 # ligne du bas
               NEXT LX
          CC = TC : LL = LL(NL) + BH     # positionner le curseur
          GOSUB curseur
          PRINT MO$(MO) ; YR             # afficher mois et année sous le cadre
          RETURN

130  PRINT HD$:PRINT TL$:FOR LX=1 TO NL:FOR VV=1 TO VS:PRINT VL$:NEXT VV:
     PRINT BL$:NEXT LX
135* CC=TC:LL=LL(NL)+BH:GOSUB800:PRINT MO$(MO) ;YR;:RETURN
```

Cette routine réalise l'affichage du cadre du calendrier.

* La version Apple de cette ligne n'en diffère que par la présence de la constante littérale " " entre les variables MO$(MO) et YR.

Figure 4.14 : Le Calendrier — Cadre

L'affichage séparé du cadre et des quantièmes (séparation rendue possible par le contrôle de position du curseur), conduit à une programmation claire et simple. L'autre solution consisterait à combiner en une routine unique, les fonctions des figures 4.14 et 4.15. Cette routine unique constituerait et afficherait en un seul passage, des lignes composites formées d'éléments du cadre et de quantièmes. Cela impliquerait que la constitution de la chaîne de caractères VL$ (faite par avance dans la routine d'initialisation) y soit intégrée. Ce qui signifie que ces trois tâches, présentées de façon claire en l'état actuel des choses, seraient alors réalisées par un programme unique et complexe, difficile à comprendre. Ceci

```
# Afficher quantièmes de 1 à ND en commençant par la SZ-ième case
affichquantiemes DY = 1
                 FOR LX = 1 TO NL
                     LL = LL(LX)                        # ligne pour la
                                                        LX-ième rangée
                     IF LX = 1 THEN
                         ZC = SZ + 1                    # ZC = 1re posi-
                                                        tion
                     else
                         ZC = 1
                     FOR CX = ZC TO 7
                         CC = CC(CX)                    # colonne pour
                                                        ce quantième
                         GOSUB curseur
                         PRINT RIGHT$(" "+ STR$(DY), 2) ; # 2 caractères,
                                                        justifiés à droite
                         DY = DY + 1                    # compter les
                                                        quantièmes affi-
                                                        chés
                         IF DY > ND THEN
                             RETURN
                         NEXT CX
                     NEXT LX
                 RETURN

400  DY=1:FOR LX=1 TO NL:LL=LL(LX):ZC=1:IF LX=1 THEN ZC=SZ+1
405  FOR CX=ZC TO 7:CC=CC(CX):GOSUB800:PRINT RIGHT$(" "+STR$(DY),2) ;
410  DY=DY+1:IF DY>ND THEN RETURN
415  NEXT CX:NEXT LX:RETURN
```

Ceci est la routine qui complète le calendrier en mettant les quantièmes dans leurs cases.

Figure 4.15 : Affichage des Quantièmes

permet de mettre en évidence l'utilité du contrôle du curseur dans la programmation des affichages.

La figure 4.16 présente la routine qui positionne le curseur en LL, CC. Sur le Pet, ceci est réalisé en faisant d'abord un PRINT du caractère " en haut ", à gauche (home) ", puis des PRINT successifs des caractères "curseur à droite " et " curseur en bas ". Cette fonction est assurée sur Apple par les instructions HTAB et VTAB. La version TRS-80 ne diffère de celle du Pet que par la valeur des caractères de contrôle du curseur et par le nombre de lignes et de colonnes de l'écran.

```
# Positionnement du curseur — l'amener en LL, CC

# Version Pet

curseur  PRINT " home "              # curseur en " haut à gauche " (home)
         CC = CC mod colonnes        # ajuster paramètres pour qu'ils soient
         LL = LL mod lignes          # dans les limites de l'écran
         IF CC > 0 THEN              # déplacement de CC vers la droite
              FOR ZZ = 1 TO CC
                   PRINT " à droite " ;
                   NEXT ZZ
         IF LL > 0 THEN              # déplacement de LL vers le bas
              FOR ZZ = 1 to LL
                   PRINT " vers le bas " ;
                   NEXT ZZ
         RETURN

800* PRINT CHR$(19);:CC=CC-40 *INT(CC/40):LL=LL-24*INT(LL/24)
805* IF CC>0THEN FOR ZZ=1 TO CC:PRINT CHR$(29);:NEXT ZZ
810* IF LL>0 THEN FOR ZZ=1 TO LL:PRINT CHR$(17);:NEXT ZZ
815  RETURN
```

C'est la routine de contrôle de position du curseur. La version Apple est plus simple car ce système comporte des commandes de déplacement du curseur.

* Sur Apple, les lignes 805 et 810 sont :

```
805  HTAB CC+1
810  VTAB LL+1
```

Sur TRS-80, les colonnes et les lignes sont au nombre de 64 et 16, et non de 40 et 24. Les caractères « en haut à gauche (home) », « à droite » et « vers le bas » sont CHR$(28), CHR$(25) et CHR$(26).

Figure 4.16 : Le Calendrier — Positionnement du curseur

```
# Initialisation spéciale pour Calendrier

calinit  READ BW, BH : DATA largeurcase, hauteurcase   # largeur de la case, hau-
                                                        teur de la case
         READ LZ, CZ : DATA lignecase, colcase          # coin supérieur gauche
                                                        de la case
         FOR XX = 1 TO 6
             LL(XX) = LZ + (XX - 1) * BH                # calculer lignes pour les
             NEXT XX                                    quantièmes

         FOR XX = 1 TO 7
             CC(XX) = CZ + (XX - 1) * BW                # calculer colonnes pour
             NEXT XX                                    les quantièmes

         TL$ = "" : BL$ = "" : VL$ = ""                 # lignes supérieure, infé-
                                                        rieure et intermédiaire
         FOR XX = 1 TO 7
             TL$ = TL$ + "quatre caractères pour ligne supérieure"
             BL$ = BL$ + "un car. coin inférieur gauche et trois car. pour ligne infé-
                                                                        rieure"
             VL$ = VL$ + "un car. pour ligne verticale et trois blancs"
             NEXT XX
         VL$ = VL$ + " un car. pour ligne verticale "
         VL$ = VL$ + " un car. pour ligne verticale "
         HD$ = " DIM LUN MAR MER JEU VEN SAM "
         VS = BH - 1                                    # nombre de lines VL$
                                                        par rangée
         TC = CZ + 5                                    # colonne de la légende
         RETURN
```

```
900*  READ BW,BH:DATA4,3
905   READ LZ,CZ:DATA3,1
910   FOR XX=1 TO 6:LL(XX)=LZ+(XX-1)*BH:NEXT XX
915   FOR XX=1 TO 7:CC(XX)=CZ+(XX-1)*BW:NEXT XX
920   TL$="":BL$="":VL$=""
925** FOR XX=1 TO 7:TL$=TL$+ "———":VL$=VL$+"|       ":BL$=BL$+"└———":
      NEXT XX
930** VL$=VL$+"| ":BL$=BL$+"| "
935   HD$="DIM LUN MAR MER JEU VEN SAM"
940   VS=BH-1:TC=CZ+7:RETURN
```

Voici l'initialisation écrite spécialement pour le Calendrier. Cette routine s'ajoute à celle de l'Anniversaire.

---

* Sur TRS-80 les deux valeurs de la ligne 900 sont 4 et 2.
** Les versions Apple et TRS-80 de ces lignes en diffèrent par l'utilisation de tirets et de points d'exclamation pour réaliser les lignes horizontales et verticales des cases du calendrier.

Figure 4.17 : Le Calendrier — Initialisation

La figure 4.17 présente la routine d'initialisation spéciale, utilisée pour le Calendrier. Cette routine est appelée en supplément de celle qui fait partie de l'Anniversaire.

En résumé

Deux jeux, l'Anniversaire et le Calendrier, ont été conçus pour illustrer des techniques importantes utilisées pour le traitement des dates en entrée et en sortie.

L'Anniversaire illustre l'algorithme qui permet de trouver le jour de la semaine correspondant à une date donnée. Ce programme donne par la même occasion, un exemple de routine de saisie de date, suffisamment souple pour être utilisée par tous.

Le Calendrier utilise les routines de l'Anniversaire avec, comme objectif, l'affichage du calendrier pour un mois et une année donnés. Ce programme montre le genre de gymnastique qui est parfois rendu nécessaire par l'adaptation d'une routine à une situation nouvelle. Il montre aussi comment on peut simplifier la structure d'un programme en utilisant les instructions de contrôle du curseur pour programmer séparément des fonctions d'affichage logiquement indépendantes.

CHAPITRE 5
Le Fisc

Je vais vous dire comment ça s'ra,
Y a un pour vous, dix-neuf pour moi,
Si cinq pour cent, c'est pas beaucoup,
Soyez heureux que j'prenne pas tout,
Car, savez vous, le Fisc, c'est moi.

— Chanson de George Harrisson

Le jeu présenté dans ce chapitre existe sous une forme ou sous une autre depuis longtemps ; mais la version décrite ici utilise probablement beaucoup mieux les possibilités de votre ordinateur individuel que ne pourraient le faire les anciennes versions conçues pour des terminaux genre télétype. Le Fisc est un jeu dont l'esprit est différent de celui des autres jeux présentés dans ce livre. L'un des buts proposés au joueur est de trouver les règles appliquées par le programme.

Explication du Fisc

Le programme commence par vous demander de fixer la taille du « gâteau ». Ce terme fait allusion aux diagrammes circulaires découpés en « parts » comme un gâteau, (tant pour le loyer, tant pour la nourriture, tant pour les impôts, etc...). La réponse à la question doit être un nombre entier.

Le gâteau montré sur l'écran ne sera pas circulaire. Ce sera un tableau rectangulaire composé de nombres. En fait, si vous entrez un nombre entier N, le gâteau sera constitué des nombres entiers 1, 2, ..., N. Le minimum autorisé pour N est 4. Le maximum dépend de la taille de votre écran et, est calculé à partir de valeurs figurant dans les instructions DATA de la routine d'initialisation (fig. 5.12).

La figure 5.1 montre un coup à titre d'exemple. Vous avez décidé que la taille du gâteau serait 15 ; les nombres 1, 2, 3, ..., 15 apparaissent d'abord sur l'écran. Tous les nombres, excepté 1, sont en « vidéo inverse ». On ne vous donne pas la raison de cette différence, mais la plupart des joueurs la découvrent assez rapidement.

Sous le tableau de la figure 5.1 apparaissent les totaux en cours :

VOUS : 0 % LE FISC : 0 % RESTE : 100 %

Au-dessous apparaît la question :

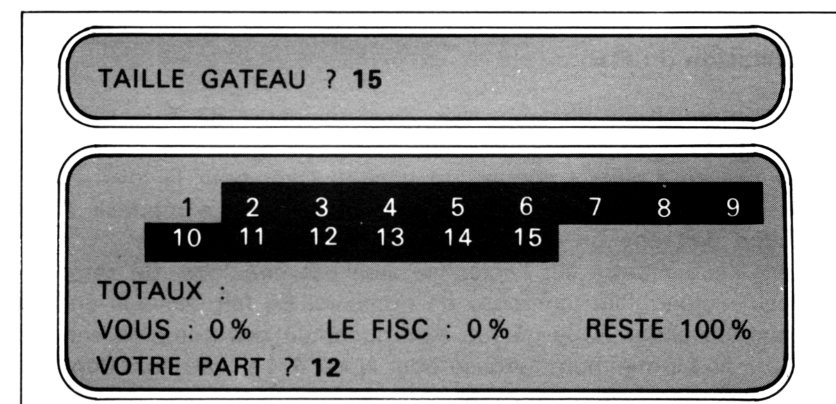
VOTRE PART ?

Cette question vous invite à entrer un nombre compris entre 1 et 15. (Cela non plus ne vous est pas indiqué, mais vous vous rendrez compte à l'usage qu'il ne faut pas entrer de nombres ne figurant pas sur l'écran.)

A la figure 5.1, vous avez entré 12. L'écran est effacé et de nouvelles informations apparaissent. Plusieurs des nombres qui constituaient le gâteau initial ont disparu et d'autres ne sont plus en **vidéo inverse**. Les nouveaux totaux apparaissent sous la représentation du gâteau :

VOUS : 10 % LE FISC : 13,3 % RESTE : 76,7 %

Il vous appartient d'expliquer comment ces nombres sont liés à la part de gâteau choisie, aux nombres manquants et à ceux qui ne sont plus en vidéo inverse. (Indication : la somme des nombres 1, 2, ..., 15 est 120.)



Au premier affichage, le joueur fixe la taille du gateau à 15. A l'affichage suivant, la totalité du gateau est montrée et le joueur choisit 12 comme part. Le troisième affichage révèle que le choix de 12 se traduit par 10 % du gâteau pour le joueur et 13,3 % pour le Fisc. Les 76,7 % restants sont représentés par un gâteau « mis à jour » ; certains nombres ont disparu, d'autres ne sont plus en vidéo inverse. Le choix de 15 comme nouvelle part par le joueur entraîne l'affichage du quatrième écran. A cet instant le joueur commet l'erreur de choisir 10 et le Fisc prend le reste du gâteau.

Figure 5.1a : Le Fisc — Exemples d'affichages

Comme nouvelle part vous avez pris 15 et l'écran montre la nouvelle situation du gâteau et les nouveaux totaux. Au coup suivant, votre choix de 10 entraîne la confiscation du reste du gâteau par le Fisc. Les totaux définitifs sont affichés à l'avant-dernière ligne :

VOUS AVEZ GARDE : 22,5 % LE FISC A PRIS : 77,5 %

Au-dessous figure un rappel de votre meilleur résultat au cours des parties précédentes :

LE RECORD EST : 54 % DE 30

Ceci signifie que lors de votre meilleure partie, vous avez fixé la taille du gâteau à 30 et que vous êtes arrivé à garder 54 % du total.

Après avoir affiché le résultat final et le meilleur score, le programme attend que le joueur entre un caractère au clavier. Il efface

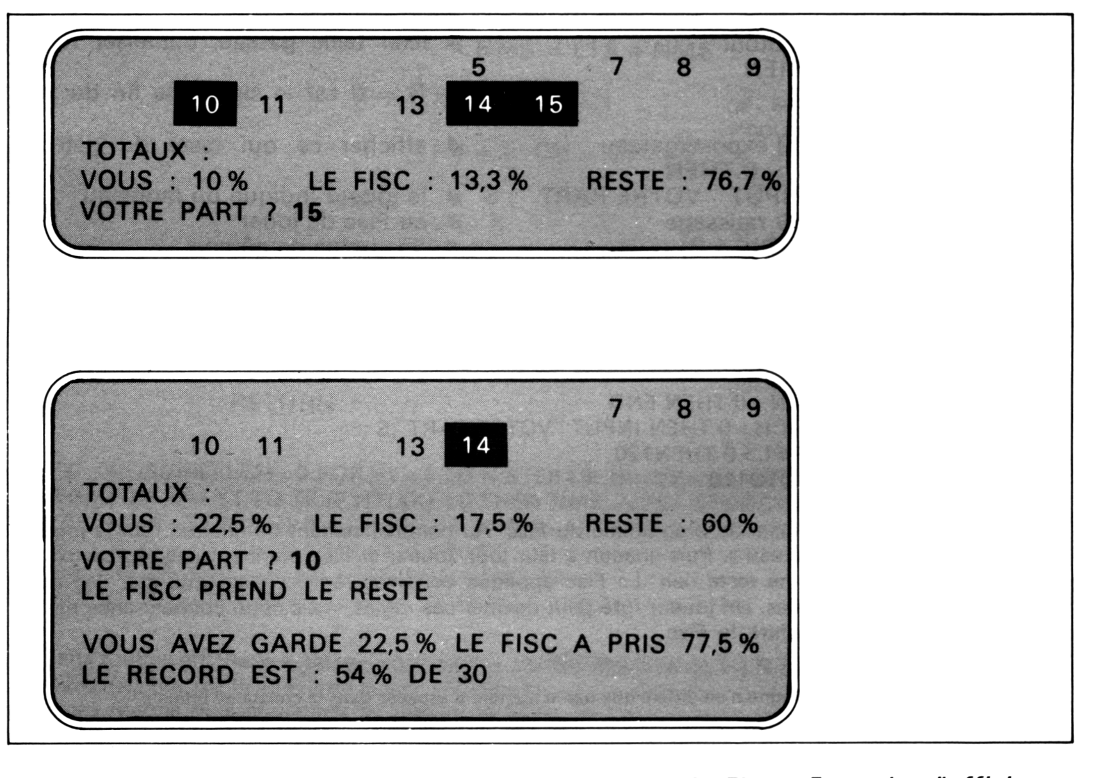


Figure 5.1b : Le Fisc — Exemples d'affichages

alors l'écran et demande qu'on fixe la taille du gâteau pour la nouvelle partie.

Le programme du Fisc

Le programme du Fisc est présenté de la figure 5.2 à la figure 5.12. Sa structure est similaire à celle des autres programmes de ce livre. La partie « secrète » du jeu figure dans les routines des figures 5.6 et 5.8. Ces routines utilisent le tableau HT pour y coder les nombres qui figurent sur l'écran, ceux qui sont en vidéo inverse, et ceux que

```
# Le Fisc
   GOSUB init                            # fixer dimensions et constantes
   repeat {
        GOSUB débutour                   # fixer taille gâteau, initialiser HT
        IF N = 0 THEN
             break                       # N = 0 est le signal de fin du jeu
        repeat {
             GOSUB exposergateau         # afficher ce qui reste du gâteau
             IF CH > 0 THEN
                  INPUT "VOTRE PART "; S # le joueur indique un morceau
             GOSUB ratissage             # au Fisc de jouer
             } until (PL = 0)            # fin si plus de gâteau
        GOSUB statis                     # afficher statistiques finales
        }
   END

100  GOSUB530
110  GOSUB420:IF N=0 THEN END
120* GOSUB150:IF CH>0 THEN INPUT"VOTRE PART";S
130  GOSUB260:IF PL>0 THEN120
140  GOSUB360:GOTO110
```

Voici la routine principale de programme du Fisc. Au commencement de chaque tour le joueur choisit une taille de « gâteau ». Puis chacun à leur tour, joueur et Fisc prennent des morceaux du gâteau jusqu'à ce qu'il ne reste rien. Le Fisc applique certaines règles, mais sans que l'on dise jamais au joueur lesquelles. Un joueur futé peut deviner ces règles, mais cette connaissance ne lui assure pas le succès contre le Fisc.

* La version Apple de cette ligne n'en diffère que par le nombre d'espaces dans la constante littérale.

Figure 5.2 : Le Fisc

Afficher ce qui reste du gâteau

```
exposergateau  GOSUB effacecran                 # démarrer sur écran effacé
               CH = 0                           # compte le nombre choix du joueur
               FOR YY = 1 TO N STEP LS          # une ligne de l'écran
                   TP = min(N, YY + LS − 1)     # dernière ligne peut ne pas être complète
                   FOR XX = YY TO TP            # nombres individuels
                       IF HT(XX) < 0 THEN
                           NM$ = L2$ + SZ$      # espace si nombre non disponible
                       else {
                           NM$ = STR$(XX)       # autrement, le nombre
                           IF XX < 10 THEN      # ajouter des blancs à gauche
                               NM$ = L2$ + NM   # pour cadrer à droite
                           else IF XX < 100 THEN  # dans une zone de 3 chiffres
                               NM$ = L1$ + NM$
                           }
                       IF HT(XX) > 0 THEN       # si l'élément du tableau «coups» différent de zéro
                           { RV = 1 : CH = CH + 1 }  # inverser vidéo, décompte choix
                       else
                           RV = 0               # pas de vidéo inverse
                       GOSUB affichnombre       # afficher le nombre
                       NEXT XX
                   PRINT                        # nouvelle ligne
                   NEXT YY
               GOSUB totaux                     # afficher totaux sous le gâteau
               RETURN
```

```
150 GOSUB520:CH=0:FOR YY=1 TO N STEP LS:TP=YY+LS−1:IF TP>N THEN TP=N
160 FOR XX=YY TO TP:IF HT(XX)<0 THEN NM$=L2$+SZ$:GOTO190
170 NM$=STR$(XX):IF XX<10 THEN NM$=L2$+NM$:GOTO190
180 IF XX<100 THEN NM$=L1$+NM$
190 RV=0:IF HT(XX)>0 THEN RV=1:CH=CH+1
200 GOSUB480:NEXT XX:PRINT:NEXT YY:GOSUB210:RETURN
```

Cette routine affiche le « gâteau » qui est la suite des nombres de 1 à N (« taille » du gâteau). Les nombres déjà pris par le joueur ou le Fisc n'apparaissent pas. Certains nombres sont en vidéo inverse pour des raisons qu'il appartient au joueur de découvrir.

***Figure 5.3 : Les Restes du Gâteau** — (Affichage)*

le joueur a le droit de choisir. La valorisation initiale de HT, les modifications qui lui sont apportées après chaque choix et la méthode de calcul des résultats sont toutes basées sur les algorithmes arithmétiques de ces deux routines.

La routine la plus compliquée est celle de la figure 5.3. C'est elle qui dessine le nouvel aspect du gâteau. L'un des objectifs du programmeur a été de rendre la routine indépendante du système BASIC utilisé. A cet effet, les techniques suivantes ont été utilisées :

— Introduction d'une variable LS (dont la valeur est précisée dans la routine d'initialisation de la figure 5.12). La valeur de LS est le nombre de chiffres figurant sur chacune des lignes représentant le gâteau. Dans l'exemple de la figure 5.1, LS vaut 9.

```
# Afficher les totaux à ce stade du jeu

totaux  PK = FNP(KP/PZ) : PK$ = LB$ + STR$(PK)   # % de revenu net
        PT = FNP(PY/PZ) : PT$ = LB$ + STR$(PT)   # % d'impôts
        PP = FNP(PL/PZ) : PP$ = LB$ + STR$(PP)   # % de gâteau disponible
        PRINT : PRINT " TOTAUX : "
        PRINT " VOUS : "; PK$ ; " % ";
             " LE FISC : "; PP$; " % ";
             " RESTE : "; PP$; " % "
        RETURN
```

```
210  PK=FNP(KP/PZ):PK$=LB$+STR$(PK)
220  PT=FNP(PY/PZ):PT$=LB$+STR$(PT)
230  PP=FNP(PL/PZ):PP$=LB$+STR$(PP)
240* PRINT:PRINT "TOTAUX:":PRINT"VOUS:";PK$;"%   LE FISC:";PT$;"%   RESTE:";
     PP$;"%"
250  RETURN
```

Cette routine affiche les pourcentages en cours des parties de gâteau appartenant au joueur, au Fisc et restant en « course ». La variable LB$ sert à s'assurer qu'un espace à gauche prend le nombre, quel que soit le système sur lequel tourne le programme. La valeur de LB$ est fixée dans la routine d'initialisation de la figure 5.12.

* La version Apple de cette ligne en diffère par le nombre d'espaces dans la constante littérale.

Figure 5.4 : Affichage des Résultats du Coup

— Les variables L1$ et L2$ qui, toutes deux, sont définies à partir de la variable LB$, sont utilisées pour la justification à droite. LB$ est définie dans la routine d'initialisation (lignes 550 et 560 de la figure 5.12) par les instructions :

```
L = LEN(STR$(1))
LB$ = " " : IF L > 1 THEN LB$ = ""
```

Ceci donne la certitude que LB$ est une chaîne de longueur nulle pour les systèmes qui font précéder d'un espace les nombres positifs (par exemple, le Pet et le TRS-80), et par contre un caractère espace lorsqu'il s'agit d'un système ne mettant rien devant ces mêmes nombres.

— Un appel à une routine séparée (GOSUB 480 à la ligne 200 de la figure 5.3) permet de passer en vidéo inverse. La routine de la ligne 480 (fig. 5.9) est spécifique d'un système donné.

Vous devriez pouvoir comprendre le reste du programme du Fisc à la seule lecture.

```
# Calcul de l'impôt
ratissage  IF CH > 0 THEN {                  # s'il y a le choix
               S = INT(S)
               IF 2 < = S < = N THEN         # et que le choix du joueur
                   IF HT(S) > = 1 THEN {     # soit valide
                       GOSUB alaloyale       # alors appliquons le règlement
                       RETURN
                       }
               }
           PRINT "LE FISC PREND LE RESTE." # autrement,...
           PY = PY + PL
           PL = 0
           RETURN

260 IF CH=0 THEN300
270 S=INT(S):IF S<2 OR S>N THEN300
280 IF HT(S)<1 THEN300
290 GOSUB310:RETURN
300 PRINT "LE FISC PREND LE RESTE.":PY=PY+PL:PL=0:RETURN
```

Cette routine décide de donner ou non ses chances au joueur. Si le Fisc ne prend pas ce qui reste, c'est un appel à la routine 5.6 qui permet de déterminer la part du Fisc.

Figure 5.5 : Calcul de l'Impôt

```
# Calculer la part du Fisc

alaloyale  TX = 0                                 # M.a.z. de la part du Fisc

           KP = KP + S : PL = PL - S              # la part du joueur est S

           HT(S) = -1                             # enlever S

           IF INT(N/S) > = 2 THEN                 # ses multiples maintenant
                FOR YY = 2 TO INT(N/S)            # tous ont un
                     HT(YY * S) = HT(YY * S) - 1  # plus petit diviseur
                     NEXT YY

           FOR XX = INT(S/2) TO 1 STEP - 1        # trouver la part du Fisc
                IF XX est un diviseur de S
                AND HT(XX) > = 0 THEN {
                     HT(XX) = - 1                 # un autre pour le Fisc
                     TX = TX + XX
                     FOR YY = 2 TO INT(N/XX)      # ajuster les diviseurs
                          HT(YY*XX) = HT(YY*XX)-1 # de ses multiples
                          NEXT YY
                     }
                NEXT XX

           PY = PY + TX : PL = PL - TX            # mise à jour de " l'impôt "
                                                  total

           RETURN
```

```
310  TX=0:KP=KP+S:PL=PL-S:HT(S)=-1:IF INT(N/S)<2 THEN330
320  FOR YY=2 TO INT(N/S):HT(YY*S)=HT(YY*S)-1:NEXT YY
330  FOR XX=INT(S/2) TO 1 STEP-1:IF HT(XX)<0 OR S<>XX*INT(S/XX) THEN350
340  HT(XX)=-1:TX=TX+XX:FOR YY=2 TO INT(N/XX):HT(YY*XX)=HT(YY*XX)-1:
     NEXT YY
350  NEXT XX:PY=PY+TX:PL=PL-TX:RETURN
```

Cette routine calcule la part de gâteau du Fisc. L'algorithme utilisé est un secret que le joueur doit s'efforcer de découvrir avant de pouvoir vraiment jouer.

Figure 5.6 : La Part du Fisc

```
# Afficher les statistiques finales

statis  PK = FNP(KP/PZ)            # « gain » en pourcentage
        PT = 100 - PK              # « impôt » en pourcentage
        IF PK > BK THEN {          # record battu ?
             BK = PK : BZ = N
             NR = 1                # oui
             }
        else
             NR = 0                # non
        PK$ = LB$ + STR$(PK)       # mettre % sous forme de chaînes
        PT$ = LB$ + STR$(PT)       # avec des blancs à gauche
        PRINT : PRINT " VOUS AVEZ GARDE "; PK$
        " % LE FISC A PRIS "; PT$ ; " % "
        IF NR = 1 THEN             # comparer avec ancien record
             PRINT " C'EST UN NOUVEAU RECORD! "
        else {
             BK$ = LB$ + STR$(BK)
             BZ$ = LB$ + STR$(BZ)
             PRINT « LE RECORD EST : »; BK$; « % DE »; BZ$
             }
        GOSUB uncar                # laisser affiché jusqu'à touche appuyée
        RETURN
```

```
360 PK=FNP(KP/PZ):PT=100-PK:NR=0:IF PK>BK THEN BK=PK:BZ=N:NR=1
370 PK$=LB$+STR$(PK):PT$=LB$+STR$(PT)
380 PRINT:PRINT "VOUS AVEZ GARDÉ";PK$;"% LE FISC A PRIS";PT$;"%"
390 IF NR=1 THEN PRINT "C'EST UN NOUVEAU RECORD!":GOTO410
400 BK$=LB$+STR$(BK):BZ$=LB$+STR$(BZ):PRINT "LE RECORD EST:";BK$"%
    OF";BZ$
410 GOSUB500:RETURN
```

Cette routine affiche les résultats finaux et les compare à ceux des tours précédents.

Figure 5.7 : Statistiques Finales

```
# Préparation du tour qui vient

débutour  GOSUB effacecran                     # démarrage sur écran propre
          repeat {
               INPUT " TAILLE GATEAU "; N      # joueur entre taille gâteau
               IF N = 0 THEN                   # N = 0, signal abandon
                    RETURN
               else
                    N = INT(N)                 # faut nombres entiers
               } until (4 < = N < = MX)
          PL = N * (N + 1)/2 : PZ = PL         # somme
          FOR XX = 1 TO N                      # effacer tableau « coups »
               HT(XX) = 0
               NEXT XX
          FOR XX = 1 TO INT(N/2)               # compter les diviseurs
               FOR YY = 2 TO INT(N/XX)
                    HT(YY * XX) = HT(YY * XX) + 1
                    NEXT YY
               NEXT XX
          KP = 0 : PY = 0                      # mise à zéro décompte gain et
                                               impôt
          RETURN
```

```
420   GOSUB520
430*  INPUT"TAILLE GATEAU";N:IF N=0 THEN RETURN
440   N=INT(N):IF N<4 OR N>MX THEN430
450   PL=N*(N+1)/2:PZ=PL:FOR XX=1 TO N:HT(XX)=0:NEXT XX
460   FOR XX=1 TO INT(N/2):FOR YY=2 TO INT(N/XX):HT(YY*XX)=HT(YY*XX)+1:
      NEXT YY:NEXT XX
470   KP=0:PY=0:RETURN
```

Cette routine permet au joueur de fixer une taille du « gâteau » à partager avec le Fisc. La routine initiatise le tableau HT, qui sert à garder trace des nombres encore disponibles et de ceux qui ne le sont pas.

* La version Apple de cette ligne n'en diffère que par le nombre des espaces dans la constante littérale.

Figure 5.8 : Préparation d'un Nouveau Gâteau

```
# Afficher NM$, en utilisant la vidéo inverse si RV = 1
# Version Pet

affichnombre  IF RV = 1 THEN
                  PRINT " inv "; NM$;" norm ";
              else
                  PRINT NM$;
              RETURN

   480*  IF RV=1 THEN PRINT CHR$(18);NM$;CHR$(146);:RETURN
   490   PRINT NM$;:RETURN
```

Voici la première de quelques routines différentes suivant le système employé. Le Pet utilise les caractères de contrôle « inv » et « norm » pour encadrer la chaîne à afficher en vidéo inverse. La version Apple est très voisine, car elle encadre une instruction PRINT NM$; entre les instructions INVERSE et NORMAL. Il faut s'y prendre d'une façon toute différente sur TRS-80 où le contrôle de vidéo n'existe pas en BASIC TRS-80.

* La version Apple de cette ligne est :

```
   480   IF RV=1 THEN INVERSE:PRINT NM$;:NORMAL:RETURN
```

Figure 5.9 : Vidéo Inverse conditionnelle

```
# Entrée mono-caractère
# Version Pet

uncar  repeat
           GET X$
           until (X$ <> "")
       RETURN

   500   GETX$:IF X$=""THEN500
   510   RETURN
```

Cette routine réalise l'entrée mono-caractère sur Pet. La même routine fonctionne sur Apple, mais la boucle n'y est plus nécessaire. Il est également souhaitable de programmer la reconnaissance du code correspondant à control-C, car l'instruction GET de l'Apple ne le fait pas. Sur TRS-80, une boucle analogue à celle ci-dessus doit être utilisée, mais en se servant de la fonction INKEY$ et non de GET (GET a une signification tout à fait différente sur un TRS-80).

Voir figure 2.11 pour les versions Apple et TRS-80.

Figure 5.10 : Entrée d'un Caractère Unique

Améliorations et additions suggérées

Voici plusieurs améliorations qu'on pourrait apporter à ce jeu :

- Utiliser le positionnement du curseur de façon à modifier l'aspect du gâteau plutôt que le réafficher en totalité, après avoir effacé l'écran.
- Garder la trace du meilleur pourcentage (et du moins bon) pour chaque taille de gâteau.
- Enregistrer les performances du joueur et les progrès réalisés. Puis, adresser des encouragements ou des félicitations à la fin de chaque tout (ou après quelques tours).

En résumé

Le Fisc est un jeu qui met le joueur au défi d'en découvrir les règles. Le programme du Fisc montre comment se rendre indépendant des petites différences existant entre les divers systèmes BASIC. Des améliorations sont possibles, particulièrement dans le domaine du contrôle du curseur et dans celui de l'enregistrement des résultats.

```
#  Effacer l'écran
#  Version Pet

effacecran   PRINT "clr";
             RETURN

  520  PRINT CHR$(147);:RETURN
```

Cette routine efface l'écran. La version Apple utilise l'instruction HOME. La version TRS-80 fait appel à l'instruction CLS.

Voir figure 2.12 pour les versions Apple et TRS-80.

Figure 5.11 : Effacement de l'Ecran

```
# Initialisation
init  READ LS : DATA tailleligne            # nombres par ligne d'écran
      READ DL : DATA lignecran              # lignes par écran
      MX = LS * (DL - textdessous)          # enlever lignes texte dessous
      DIM HT(MX)                            # dimensionner HT d'après
      L = LEN(STR$(1))                      # voir si BASIC met blanc à gauche
      SZ$ = ""                              # SZ$ = un nombre de 1 chiffre à
                                            blanc
      FOR LL = 1 TO L
           SZ$ = SZ$ + " "
           NEXT LL
      IF L > 1 THEN                         # LB$ garantit un blanc à gauche
           LB$ = ""                         # LB$ est « vide » si BASIC met un
                                            blanc
      else
           LB$ = " "
      L1$ = LB$ + " "                       # blancs à gauche en plus pour
                                            nombre de deux chiffres
      L2 $ = L1$ + " "                      # pour nombre de 1 chiffre
      DEF FNP(X) = INT(1000 * X + .5)/10    # fonction pour XX.X %
      RETURN

530* READ LS:DATA 9:READ DL:DATA 24
540  MX=LS*(DL-5):DIM HT(MX)
550  L=LEN(STR$(1)):SZ$="":FOR LL=1 TO L:SZ$=SZ$+" ":NEXT LL
560  LB$=" ":IF L>1 THEN LB$=""
570  L1$=LB$+" ":L2$=L1$+""
580  DEF FNP(X)=INT(1000*X+.5)/10
590  RETURN
```

Ceci est la routine d'initialisation du programme du Fisc. Les variables SZ$, LB$, L1$, L2$ sont « valorisées » à l'aide d'un programme unique, mais les valeurs affectées seront différentes. Ceci est un exemple de programmation indépendante du système.

* Sur TRS-80, les valeurs de LS et DL sont 15 et 16.

Figure 5.12 : Initialisation

CHAPITRE 6

Le Basic Libre

Dans les premiers chapitres, vous avez vu de nombreux exemples de descriptions de programmes en BASIC Libre. Chaque programme y a été présenté sous deux formes : sa description en BASIC Libre et le listing des instructions BASIC réelles. Nos commentaires ont jusqu'à présent, fait référence à cette dernière forme. Les règles d'utilisation du BASIC Libre vont maintenant être expliquées. Il vous faudra connaître ces règles si vous désirez aller plus avant dans ce livre, car les analyses des différents programmes ne se référeront plus aux instructions BASIC réelles. Désormais, il ne sera fait référence qu'au BASIC Libre. Nous allons commencer par étudier le processus de conception d'un programme.

Programmer en BASIC est simple — mais simple d'une façon trompeuse. C'est comme apprendre à planter des clous et à scier du bois ; vous pouvez acquérir ces techniques rapidement, elles ne suffiront pas à faire de vous un charpentier. Un programmeur de qualité, comme le bon charpentier, doit faire beaucoup plus que se servir de façon correcte des outils qui sont à sa disposition. L'artisan qualifié — charpentier ou programmeur — s'appuie toujours sur la réflexion et une préparation soignée. S'asseoir devant un clavier et commencer à écrire un programme revient à construire une maison sans en avoir les plans.

Techniques de conception des programmes

Les trois principaux outils dont les programmeurs disposent pour préparer et décrire leurs programmes sont :

— Les organigrammes.

— Les descriptions d'algorithmes en langage usuel.

— Les pseudocodes.

Pour illustrer ces techniques, nous allons utiliser le programme de l'Entraînement à l'Addition du Chapitre 1.

Puisque probablement, vous connaissez déjà les organigrammes, nous ne verrons pas cette technique dans le détail. La figure 1.1 montre l'organigramme du programme de l'Entraînement à l'Addition qui se trouve en figure 1.2.

Une description d'algorithme en langage usuel est un organigramme sans dessins. Les phrases, au lieu d'être mises dans des cases

deviennent des « étapes » et les flèches de l'organigramme sont remplacées par une numérotation des étapes, comme ci-après :

" Retournez à l'étape 39 "

La figure 6.1 montre une description d'algorithme en langage usuel qui correspond à l'organigramme de l'Entraînement à l'Addition.

La figure 6.2 montre le pseudocode qu'un programmeur pourrait avoir rédigé avant d'écrire le programme d'Entraînement à l'Addition. Bien que le contenu du pseudocode soit le même que celui de l'organigramme et de la description en langage usuel, la forme est légèrement différente. Tout d'abord, la boucle du programme est indiquée par le mot « repeat » suivi de l'indication des actions qui doivent être répétées. Ces actions sont délimitées par des accolades. Notez que la version en pseudocode ne contient ni flèches ni numéros d'étapes. Dans cette dernière représentation, il n'y a pas d'équivalent de l'instruction GOTO du BASIC. Les boucles et les branchements qui ne peuvent être réalisés par des boucles FOR ... NEXT, le sont au moyen de « super-instructions » comme « repeat ». De ce fait, GOTO n'est pas nécessaire dans une représentation de type pseudocode.

L'expérience de langages « structurés » comme Pascal et le langage C, démontre que cette façon de concevoir les programmes, connue sous le nom de *programmation sans GOTO*, conduit à des programmes qui sont plus faciles à écrire, à mettre au point et à comprendre que ceux qui sont construits avec des instructions GOTO.

1. Choisir deux nombres à 1 seul chiffre, N1 et N2.
2. Demander « Que font N1 + N2 ? ».
3. Si la réponse du joueur est exacte, sauter à l'étape 6.
4. Annoncer « C'est faux, recommencez. »
5. Revenir à l'étape 2.
6. Annoncer « Exact, essayez en un autre. »
7. Revenir à l'étape 1.

Voici une description verbale (en langage usuel) du programme Entraînement à l'Addition présenté au chapitre 1. La séquence des opérations commence à l'étape 1 et continue dans l'ordre croissant des étapes, sauf quand on rencontre une étape (comme 3, 5 ou 7 dans l'exemple ci-dessus) qui modifie le déroulement en séquence de façon explicite.

Figure 6.1 : L'Entraînement à l'Addition — Description d'algorithme en langage usuel

Le BASIC Libre

Le BASIC Libre est une forme de pseudocode. Il est beaucoup plus précis que le pseudocode de la figure 6.2, car il utilise des instructions BASIC réelles de préférence au langage courant pour la description des étapes.

Le BASIC Libre a été développé par l'auteur pour atteindre deux objectifs principaux :

— Faire en sorte que le programmeur soit libéré des numéros de lignes du BASIC.

— Permettre l'application au BASIC, des techniques de programmation sans GOTO.

La figure 6.3 montre comment le BASIC Libre a été utilisé dans le présent ouvrage. A chaque figure, la description en BASIC Libre est donnée en premier. A la figure 6.3, la description concerne un programme très simple ayant la structure d'une boucle sans fin. Le programme lit les valeurs des diamètres dans l'instruction DATA et affiche les valeurs des circonférences correspondantes.

Sous la description en BASIC Libre, se trouvent les instructions BASIC effectives, écrites à partir du BASIC Libre. La traduction du BASIC Libre en BASIC a été faite manuellement en utilisant des règles simples et « mécaniques ». Dans la plupart des cas, deux programmeurs différents partant d'une même description en BASIC Libre écrivent des programmes BASIC identiques.

```
repeat {
     choisir deux nombres d'un seul chiffre N1 et N2
     repeat {
          demander " Que font N1 + N2 ? "
          « if » la réponse est fausse
               annoncer " Faux, essayez de nouveau "
          else
               annoncer " Exact, essayez-en un autre "
          } until (réponse soit correcte)
     }
```

Voici la version « pseudocode » du programme de l'Entraînement à l'Addition. Les « étapes » sont les mêmes que celles de l'organigramme (fig. 1.1) ou de l'algorithme verbal (fig. 6.1), mais l'enchaînement des opérations est présenté d'une façon tout à fait différente.

Figure 6.2 : L'Entraînement à l'Addition — Pseudocode

Il a été prévu que le BASIC Libre serait traduit en BASIC de façon manuelle. Par conséquent, on n'y a pas inclus les mécanismes formels qu'aurait nécessité un processus informatisé de traduction. En examinant le programme de la figure 6.3 plus en détail, nous verrons des exemples de souplesse du BASIC Libre qui rendent la conversion par ordinateur impossible.

Nous allons maintenant étudier les règles d'écriture du BASIC Libre.

Utilisation des lettres minuscules

La première chose à noter est qu'en BASIC Libre, les minuscules servent à représenter les éléments qui n'appartiennent pas au langage BASIC et que par contre, les majuscules sont utilisées pour tout ce qui pourra être transcrit tel quel dans le programme BASIC. Par

```
repeat {
      READ D
      C = pi * D
      PRINT "DIAMETRE : "; D ; " CIRCONFERENCE : "; C
      }
DATA 4, 6, 10
```

```
10  READ D
20  C=3.14159*D
30  PRINT "DIAMETRE:";D;"CIRCONFERENCE:";C
40  GOTO 10
50  DATA 4,6,10
```

Cette figure présente la description en BASIC Libre d'un programme très simple. Sous le BASIC Libre, se trouve le jeu d'instructions en BASIC Réel transcrites à partir du BASIC Libre.La comparaison des versions BASIC et BASIC Libre met en évidence plusieurs points concernant cette dernière :

- Le BASIC Libre est « sans GOTO » et n'utilise pas de numéros de ligne.
- L'indentation du texte est utilisé en BASIC Libre pour renseigner le lecteur sur la structure du programme.
- Les lettres minuscules forment les termes du BASIC Libre qui n'appartiennent pas au BASIC Réel.
- Les Paramètres, c'est-à-dire les constantes à nom symbolique, sont utilisés en BASIC Libre, mais aucune indication quand aux valeurs réelles qu'il conviendra de leur affecter, n'apparaît en BASIC Libre.

Figure 6.3 : Un Programme Simple en BASIC Libre

exemple, à la figure 6.3 les mots « repeat » et « pi » sont écrits en minuscules. Le mot « repeat » est l'une des instructions de contrôle du BASIC Libre. (Nous étudierons bientôt toutes les structures de contrôle du BASIC Libre.) Le mot « pi » est un exemple de la souplesse du BASIC Libre : c'est un nom symbolique pour la constante qui apparaîtra dans le programme BASIC (3,14159 à la figure 6.3). Cependant, nulle part dans la description en BASIC Libre, nous n'avons écrit la définition « pi : = 3,14159 ». Cette définition permettrait à un ordinateur convenablement programmé, de produire le programme BASIC à partir de la description en BASIC Libre de la figure 6.3. Aucune règle ne codifie ces définitions, car ce surcroît de formalisme n'ajouterait rien à l'utilité du BASIC Libre dans ce livre.

Une constante qui a reçu un nom symbolique, est un *paramètre* du programme. Cette appellation est tirée du vocabulaire de la statistique mathématique. Un paramètre dans une distribution statistique est une constante qui figure dans la formule de calcul de cette distribution. Des valeurs différentes de cette constante conduisent à des distributions différentes mais apparentées. D'une façon similaire, dans le programme de la figure 6.3, des valeurs différentes du paramètre « pi » donnent des programmes différents mais apparentés. Une valeur de 3,14 produira une série de circonférences dont la valeur différera de celles obtenues à partir de 3,14159.

Indentation

Notons que la description de programme en BASIC Libre de la figure 6.3 utilise l'indentation, c'est-à-dire le décalage des instructions, pour refléter la structure du programme. Le programme comprend deux instructions : l'instruction repeat et l'instruction DATA. C'est pourquoi les mots « repeat » et « DATA » sont les seuls mots écrits à l'extrême-gauche. Les trois instructions comprises dans l'instruction repeat sont décalées à partir de cette position ; si l'une de ces instructions contenait elle-même d'autres instructions, ces dernières donneraient lieu à un niveau d'indentation supplémentaire. Remarquons que l'accolade de fermeture à droite est seule sur sa ligne, au même niveau que la séquence d'instructions qu'elle délimite.

Ces règles d'indentation ont pour objet de rendre plus lisible la description. Mais, en réalité, le fond même de la description ne changerait pas s'il n'y avait aucun décalage ou si les accolades étaient placées sur la même ligne que les instructions qu'elles délimitent.

Commentaires

La plupart des langages de programmation donnent au programmeur la possibilité d'inclure des commentaires dans le code source de son programme. Le commentaire est un « aparté » destiné au lecteur pour préciser le but ou la signification des instructions. Le BASIC Libre étant la forme sous laquelle le programme BASIC est destiné à être lu et compris, il est essentiel que des commentaires puissent y figurer. En BASIC Libre, tout texte entre un « # » et la fin de la ligne, est un commentaire. On en trouve des exemples dans tous les programmes de jeux de ce livre.

Traduction du BASIC Libre en BASIC

Il n'existe pas de programme qui transforme le BASIC Libre en BASIC. Il faut donc que vous assuriez la traduction vous-même. Regardons les règles du BASIC Libre et voyons quelles sont les techniques de traduction disponibles.

L'instruction Repeat

La forme générale de l'instruction repeat est

```
repeat instruction until (condition)
```

Par exemple,

```
repeat X = X + 1 until (X = 9)
```

deviendrait en BASIC

```
10  X = X + 1
    IF X <> 9 THEN 10
```

D'une façon générale, la signification de

```
repeat instruction until (condition)
```

est : répéter l'exécution de l'instruction, en faisant suivre chaque exécution d'un test de condition. Si la condition est vraie, on cesse d'exécuter l'instruction. Quelles que soient l'instruction et la condition, la forme générale de l'ordre repeat peut être convertie en BASIC par les instructions suivantes :

```
N  instruction
   IF NOT condition THEN N
```

où N est le premier numéro de ligne disponible. Si votre ordinateur individuel ne reconnaît pas l'opérateur logique NOT (tous les principaux ordinateurs individuels le font) ou si vous voulez optimiser le programme BASIC, vous pourrez transformer « NOT condition » en une condition équivalente. Dans l'exemple donné plus haut, nous avons écrit « X <> 9 » au lieu de « NOT X = 9. »

La définition générale que nous venons de donner, ne permet de répéter qu'une seule instruction. Les accolades permettent d'étendre cette structure à la répétition de nombreuses instructions. Les accolades jouent, en BASIC Libre, le rôle des parenthèses dans les expressions arithmétiques. Par exemple, dans une expression arithmétique, si nous voulons multiplier un terme par 5, nous pouvons écrire

5 * term

Si le terme en question est la variable A, nous pouvons écrire

5 * A

mais si le terme à multiplier par 5 est A + B + C, nous devrons écrire

5 * (A + B + C)

Les parenthèses ont pour effet que ce qui y est inclus, sera traité comme un terme unique. D'une façon analogue, quand nous mettons des instructions entre accolades, comme dans la figure 6.3, tout ce qui est à l'intérieur sera traité comme s'il s'agissait d'une seule instruction de la description en BASIC Libre.

L'instruction repeat de la figure 6.3 ne comporte pas la partie « until (condition) ». La figure 1.3 montre une description en BASIC Libre du programme d'Entraînement à l'Addition. L'instruction

```
150   IF A <> N1 + N2 THEN ... GOTO 120
```

de la figure 1.2 correspond à la partie

until (A = N1 + N2)

de l'instruction repeat intérieure de la figure 1.3.

Il convient de noter que la condition qui suit le mot « until » est toujours entre parenthèses. On sait comme cela, de quoi se compose exactement la condition. En fait, dans la plupart des cas, il n'y aurait probablement pas d'ambiguïté, mais la mise entre parenthèses rend la description du programme plus claire et ceci notamment lorsque la condition comporte plusieurs éléments. Les conditions qui suivent « until » dans une instruction repeat sont identiques à celles des

instructions IF, mais dans ces dernières, on n'utilise pas de parenthèses pour encadrer la condition, parce que le BASIC sait que celle-ci est toujours constituée par ce qui est compris entre le mot IF et le mot THEN.

Il existe une autre façon de terminer une instruction repeat, en utilisant l'instruction break. Celle-ci arrêtera la répétition de l'instruction dont elle est un des éléments. Un exemple de l'utilisation de l'instruction break figure dans la structure générale (routine principale) du programme du Jumelage (fig. 7.16, chap. 7). Souvent son utilisation peut être évitée, en restructurant le programme. Cette réorganisation sera souvent bénéfique, car le recours à l'instruction break peut être le signe d'une mauvaise organisation du programme.

IF ... THEN ... else

Il y a plus de 2 000 ans, Aristote énonçait les principes de la logique qui sont devenus l'outil principal de raisonnement de la pensée occidentale. L'un d'entre eux qui est le *principe du tiers exclu*, dit : « Toute affirmation est vraie ou fausse — elle doit être l'une ou l'autre et ne peut être les deux. » Cette règle a servi de modèle à la structure IF ... THEN ... else du BASIC Libre. La forme générale de la structure est :

IF condition THEN instruction 1 else instruction 2

La partie « else instruction 2 » est facultative. Cette instruction a l'effet suivant : si la « condition » est vraie, alors l'« instruction 1 » est exécutée ; si la « condition » est fausse, alors l'« instruction 2 » est exécutée. Il en résulte qu'une et une seule des instructions est exécutée. (Si la « condition » est fausse et que la partie « else instruction 2 » est omise, alors rien n'est exécuté.) Comme avec l'instruction « repeat », l'« instruction 1 » et l'« instruction 2 » peuvent être des instructions isolées ou des groupes d'instructions délimitées par des accolades.

La figure 1.3 qui présente une description en BASIC Libre du programme d'Entraînement à l'Addition, donne un exemple de cette structure et des règles que nous suivrons pour l'indentation et le regroupement de ses différentes parties. Dans cette figure la « condition » est « A = N1 + N2 », l'« instruction 1 » est « PRINT " C'EST EXACT etc. "» et l'« instruction 2 » est « PRINT " C'EST FAUX etc. "».

Il convient de noter les règles d'utilisation des majuscules et des minuscules dans la structure IF ... THEN ... else. Nous aurions pu

écrire if ... then ... else (entièrement en minuscules) pour distinguer la structure BASIC Libre du IF ... THEN du BASIC Réel. Mais, le IF ... THEN dans le IF ... THEN ... else du BASIC Libre se traduit toujours en BASIC par IF ... THEN. La syntaxe du BASIC Libre met l'accent sur ce point en écrivant IF ... THEN en majuscules. Etant donné que la plupart des systèmes BASIC sur ordinateurs individuels ne permettent pas d'utiliser ELSE avec IF ... THEN (le TRS-80 a le ELSE, le Pet et l'Apple ne l'ont pas), nous avons utilisé des minuscules pour écrire le mot else.

L'Ordre Case

L'ordre « case », qui est une forme très utile de l'ordre IF, est fondée sur une autre des méthodes favorites d'Aristote, l'énumération des cas possibles. La forme générale en est :

```
IF case
     condition 1 THEN instruction 1
     condition 2 THEN instruction 2
                    .
                    .
                    .
     condition n THEN instruction n
else
     instruction n + 1
```

La partie « else instruction n + 1 » de cette structure est facultative.

L'ordre « case » contribue grandement à la clarté des programmes. Quand on regarde la structure ci-dessus, on voit qu'une et une seulement des instructions qui en font partie, sera exécutée. L'instruction exécutée sera celle qui correspond à la première condition vérifiée. Si toutes les conditions sont fausses, c'est l'instruction n + 1 qui sera exécutée.

La figure 6.4 montre un programme en BASIC Libre qui utilise un ordre « case ». A l'intérieur de cet ordre « case », on trouve trois conditions et une partie else. Ce qui fait le reste de ce programme de jeu, nous vous laissons le soin de le découvrir. En fait, le but même du jeu (appelé « Commentez-le, Si Vous Pouvez »), est de trouver ce que fait le programme. La figure 6.5 présente le programme BASIC obtenu à partir de la figure 6.4. Si vous avez travaillé comme programmeur, si vos amis vous ont envoyé des programmes de jeux ou si encore vous les avez tirés de livres ou de revues, il est alors vraisemblable que vous avez déjà eu à déchiffrer des programmes du genre de « Commentez-Le, Si Vous Pouvez ».

```
INPUT A$, B$

IF case

    A$ = " A " THEN
        IF B$ = " 1 " THEN
            PRINT " A1 "
        else
            PRINT " AX "

    A$ = " B " THEN
        repeat
            INPUT C$
            until (B$ = C$)

    A$ = " C " THEN
        IF VAL(B$) < = O THEN
            PRINT " ? ? ? "
        else {
            FOR J = 1 TO VAL(B$)
                PRINT "*";
                NEXT J
            repeat {
                INPUT C$
                IF C$ = " Q " THEN
                    break
                else
                    PRINT " XYZ "
                }
            }

    else
        PRINT " HA HA "

END
```

Ce programme réalise un jeu appelé « Documentez-le, Si Vous Pouvez ». Le but du jeu est de trouver ce que le jeu peut bien faire. Vous avez le droit de faire tourner le programme et de lire le listing (le niveau au-dessus du jeu consiste à lire le listing en BASIC Réel mais non celui en BASIC Libre.) Acquérir une certaine force à ce jeu peut s'avérer utile dans des situations de la vie réelle.

Figure 6.4 : Un Programme Qui Utilise un Ordre Case

While

La dernière structure de contrôle dont nous allons parler, est utilisée pour créer des boucles commençant par un test. La forme générale de la structure « while » est :

while (condition) instruction

Dans une structure « while », l'instruction ne sera pas exécutée si dès le départ, la condition est fausse. (La figure 4.6 du chapitre 4 montre un programme qui utilise une structure « while. »)

Sous-programmes

Le principal objectif du BASIC Libre était de libérer le programmeur d'avoir à penser aux numéros de lignes pendant la conception du programme BASIC. Les structures de contrôle que nous venons de présenter, suppriment les GOTO et les numéros de lignes dans les ordres IF. Il reste cependant un problème : comment éliminer les numéros de lignes dans les instructions GOSUB. La solution que nous avons retenue, consiste à désigner la première ligne de chaque sous-programme par référence symbolique. (Si vous pensez qu'une instruction GOTO est nécessaire dans votre programme, vous pouvez également référencer cette ligne de façon symbolique.) Ces références symboliques sont attribuées en écrivant un nom (en minuscules, bien sûr) sur la gauche de l'instruction exactement à la place qu'occuperait un numéro de ligne.

Traduction « Mécanique »

D'un bout à l'autre de ce livre, figurent des programmes pour lesquels sont données la description en BASIC Libre et la liste des instructions BASIC réelles. Ces programmes sont des exemples de mise en pratique de la traduction du BASIC Libre en BASIC. Dans de nombreux cas, la traduction manuelle a été meilleure que celle qui aurait été produite par un logiciel de bonne qualité. Pour chacune des structures présentées dans ce chapitre, il existe une traduction « évidente » en BASIC. La figure 6.6 récapitule ces correspondances.

Le BASIC Libre, la programmation structurée et Pascal

Le BASIC Libre est un BASIC « structuré ». Vous avez déjà certainement entendu beaucoup de choses à propos de la « pro-

grammation structurée » (structured programming). Ce terme a son origine dans les « Notes on Structured Programming » de Edsger W. Dijkstra (éditées dans « Structured Programming » de Dahl *et al.*; Academic Press, 1972). Dans cette monographie, Dijkstra met en évidence deux choses :

— Les programmes sans GOTO sont plus faciles à comprendre que les programmes qui en contiennent, et leur exactitude est plus facile à prouver.

— On peut obtenir des améliorations substantielles en réorganisant les programmes en « couches », chacune de ces couches représentant un niveau différent d'abstraction et utilisant une machine « idéale » adaptée à ce niveau.

Le premier de ces points est le plus facile à comprendre. C'est d'ailleurs pourquoi de nombreux livres, articles et séminaires ont eu pour objectif d'enseigner la « programmation structurée », mais en réalité se sont essentiellement consacrés aux techniques de la programmation sans GOTO. Un des objectifs majeurs du BASIC Libre est de mettre à la disposition des programmeurs BASIC les avantages de la programmation sans GOTO ; on peut dire qu'à cet égard, le BASIC Libre met en œuvre les techniques de la programmation structurée.

```
 10  INPUT A$,B$
 20  IF A$<>"A" THEN 40
 30  IF B$="1" THEN PRINT "A1":GOTO 110 ELSE PRINT "AX":GOTO 110
 40  IF A$<>"B" THEN 60
 50  INPUT C$:IF B$<>C$ THEN 50 ELSE 110
 60  IF A$<>"C" THEN 100
 70  IF VAL(B$)<=0 THEN PRINT "???":GOTO 110
 80  FOR J=1 TO VAL(B$):PRINT "*";:NEXT J
 90  INPUT C$:IF C$="Q" THEN 110 ELSE PRINT "XYZ":GOTO 90
100  PRINT "HA HA"
110  END
```

C'est le jeu « Documentez-le, Si Vous Pouvez » présenté à la figure 6.4.Il est écrit en BASIC TRS-80 qui permet l'utilisation de Clauses ELSE dans les instructions IF. Le programme serait plus compliqué sur Apple ou Pet, car ces systèmes n'autorisent pas la clause ELSE.

Figure 6.5 : Le Jeu « Documentez-Le, Si Vous Pouvez » BASIC Réel

BASIC Libre		BASIC
repeat	nnn	instruction
instruction		IF NOT condition THEN nnn
until (condition)		
while (condition)	nnn	IF NOT condition THEN mmm
instruction		instruction
		GOTO nnn
	mmm	...
IF condition THEN		IF NOT condition THEN nnn
instruction 1		instruction 1
else		GOTO mmm
instruction 2	nnn	instruction 2
	mmm	...
IF case		
condition 1 THEN instruction 1		IF NOT condition 1 THEN nn2
.		instruction 1
.		GOTO mmm
condition 2 THEN instruction 2	nn2	IF NOT condition 2 THEN nn3
		instruction 2
.		GOTO mmm
		.
.		.
		.
condition n THEN instruction n	nnn	IF NOT condition n THEN eee
		instruction n
		GOTO mmm
else	eee	instruction n + 1
instruction n + 1	mmm	...

On met en évidence ici une correspondance simple entre les structures de contrôle du BASIC Libre et des instructions BASIC. Un logiciel convenable pourrait réaliser cette transformation.

Figure 6.6 : Correspondance entre BASIC Libre et BASIC Réel

Le second point est plus difficile à appréhender et il est difficile d'apprendre à pratiquer la technique recommandée. Les programmes de ce livre n'ont pas été conçus par un effort conscient de développement par « couches », ou par la définition de machines abstraites « idéales » ; à cet égard, ils ne sont pas des exemples de programmation structurée. Ou plutôt, c'est la technique voisine de la démarche « descendante » qui a été utilisée. Les techniques de la structuration descendante sont discutées dans plusieurs chapitres. (Par exemple, au chapitre 7, Jeu du Jumelage.)

Pascal est un langage de programmation conçu pour faciliter la programmation sans GOTO. Les structures de contrôle de Pascal sont essentiellement les mêmes que celles qui sont utilisées en BASIC Libre, de telle sorte qu'une description de programme en BASIC Libre ressemble à un langage hybride qu'on aurait obtenu par croisement de BASIC et de Pascal. De ce fait, de nombreuses descriptions de programmes en BASIC Libre figurant dans ce livre pourraient servir de base à l'écriture de programmes Pascal. Il y a, cependant, deux obstacles majeurs à une telle traduction :

— Certaines des possibilités de BASIC n'existent pas en Pascal « standard ».

— Les programmes Pascal ont besoin que toutes les données et sous-programmes soient « déclarés ». Pascal impose de déclarer le type de chaque objet avant de l'utiliser et s'assure ensuite que l'usage qui en est fait, est bien compatible avec ce type.

En résumé

Trois techniques principales sont utilisées par les programmeurs pour préparer et décrire leurs programmes : les organigrammes, les descriptions d'algorithmes en langage usuel et le pseudocode. Le BASIC Libre est une forme de pseudocode si précise qu'il s'agit presque d'un langage de programmation. Le BASIC Libre attribue par convention des significations différentes aux lettres majuscules et minuscules. Les majuscules sont utilisées pour écrire les parties du BASIC Libre qui réapparaîtront telles quelles dans le programme BASIC résultant. Les minuscules sont utilisées partout ailleurs dans la description en BASIC Libre. Les principales utilisations des minuscules sont les paramètres du programme et les mots-clés identifiant les différentes structures de contrôle. L'indentation est utilisée dans les descriptions en BASIC Libre pour refléter la structure du programme. L'utilisation de cette dernière règle est facultative

et les programmeurs peuvent adopter des styles d'indentation différents de celui de l'auteur.

Les structures de contrôle utilisées en BASIC Libre sont :

— repeat ... until
— IF ... THEN ... else
— IF case ... else
— while

L'instruction « break » permet de sortir de façon « imprévue » des structures « repeat » et « while ».

Ces caractéristiques rendent inutiles le recours aux GOTO dans les descriptions en BASIC Libre. De plus, des « numéros » de lignes symboliques sont utilisés pour appeler les sous-programmes par GOSUB. C'est là l'étape finale qui permet au BASIC Libre d'atteindre son principal objectif, libérer le programmeur du carcan des numéros de ligne.

Le BASIC Libre assure l'aspect « programmation sans GOTO » de la programmation structurée. Ses structures de contrôle sont les mêmes que celles de Pascal, de sorte qu'une description en BASIC Libre peut être, sans difficulté majeure, codée en Pascal. Les principaux obstacles rencontrés par ces traductions tiennent d'une part, au fait que Pascal n'a pas certaines des possibilités de BASIC et d'autre part, à ce qu'il exige la « déclaration » des données et des sous-programmes.

CHAPITRE 7

Le Jeu du Jumelage

Le chapitre présente le Jeu du Jumelage. C'est un jeu complexe qui éclaire des points importants ayant trait à la programmation des systèmes interactifs. Nous commencerons par décrire la manière d'y jouer ; nous présenterons ensuite les programmes correspondants. L'objet du jeu du Jumelage est de « jumeler » des éléments compatibles en provenance de deux groupes différents. Ces deux groupes peuvent être des hommes et des femmes, des vendeurs et des acheteurs, des animaux familiers et des futurs acquéreurs, des emplois et des personnes à la recherche d'un emploi et ainsi de suite. La compatibilité est mesurée de la façon suivante : on établit deux séries de questions, et les réponses de chaque joueur sont ensuite comparées aux préférences « affichées » par les membres de l'autre groupe.

Si, par exemple, les deux groupes concernés sont les hommes et les femmes qui assistent à une réunion, le jeu consistera à trouver les couples assortis. Pour ceci, on commence par donner des noms à chacun des deux groupes (LES HOMMES, LES FEMMES). Ensuite, on établit une série de questions à choix multiples pour chaque groupe, un coefficient (ou poids) étant attribué à chaque question. Ce coefficient sert à déterminer à quel point les réponses d'un joueur donné correspondent aux préférences affichées par les membres de l'autre groupe.

Regardons comment se déroule une partie. Celle-ci commence par l'apparition du signal " : : " dans le coin supérieur gauche de l'écran. La figure 7.1 montre quelles sont les commandes que l'on peut entrer en réponse au signal " : : ". Chaque commande est constituée par une lettre ou un chiffre unique.

La phase préparatoire

La première phase dite « préparatoire » est celle qui consiste à saisir les noms des groupes et les questions à choix multiples. La figure 7.2 donne un exemple du dialogue qui pourrait suivre l'entrée au clavier de la commande " I ". Toutes les réponses du joueur doivent être conclues par RETURN (ou ENTER). Les réponses du joueur sont en caractères gras pour les distinguer des questions posées et signaux affichés par l'ordinateur.

A la figure 7.2, on voit que le joueur a tapé I et que le programme a répondu par le message " GROUPE 1 : " pour indiquer au joueur

qu'il doit maintenant entrer le nom du premier groupe. Le joueur tape " LES HOMMES " et le programme répond immédiatement par " Q1 POUR LES HOMMES " pour demander les différents éléments de la première question destinée au groupe 1. Si le joueur avait fourni " LES LOPINCHIENS " comme nom du groupe 1, le programme aurait alors demandé " Q1 POUR LES LOPINCHIENS ".

Comme première question destinée aux HOMMES, le joueur a entré " POUR QUI AVEZ-VOUS LE PLUS DE RESPECT ? ". Notons que le joueur a pu taper ce libellé qui contient des blancs, sans le mettre entre guillemets. Cela laisse supposer que l'instruction LINE INPUT est utilisée. Ultérieurement, quand nous étudierons le programme, nous verrons que LINE INPUT est simulée par une routine.

Puis, le programme envoie comme signaux d'appel, des nombres, et le joueur répond en indiquant quels sont les choix possibles pour la première question. Dans notre exemple, après que cinq options aient été saisies, le programme n'appelle pas la suivante et demande : " COEFFICIENTS : ". Cela signale au joueur qu'il doit indiquer le « poids » attribué à la question. A la figure 7.2, le programme cesse de passer à l'option suivante lorsque les 5 premières ont été saisies, parce que le nombre limite d'options par question a été fixé à 5, par une instruction DATA dans la routine d'initialisation.

Nous verrons ultérieurement quelles valeurs il faut donner aux paramètres du programme. Mais, notons dès maintenant, que si vous avez un petit ordinateur individuel (par exemple, avec seulement 8K de mémoire vive), vous serez peut-être obligé de donner des valeurs faibles au nombre maximum d'options par question, au nombre de questions et au nombre de joueurs, parce que ces paramètres servent à définir la taille de tableaux à plusieurs dimensions.

Après que le joueur ait fixé à 4, le « poids » de la première question, le programme demande " Q2 POUR LES HOMMES " et le joueur répond en donnant le libellé de la question suivante : " QUE DESIREZ-VOUS LE PLUS DANS LA VIE ? ". A nouveau, le programme en affichant les numéros des options, invite le joueur à indiquer quelles sont les différentes réponses possibles. Mais cette fois-ci, après avoir entré trois options, le joueur a frappé la lettre " Q " en réponse au signal numéro quatre.

Tout au long du jeu du Jumelage, la lettre Q est utilisée comme signal de « fin » ou d'« abandon ». Dans le cas présent, la signification en est : « J'en ai terminé avec les options possibles prévues pour cette question. Continuons avec la donnée suivante (le coefficient de la question). » Le programme enchaîne donc en affichant " POIDS : ", et le joueur indique comme valeur, 6. Le programme affiche alors le signal " Q3 POUR LES HOMMES ". Le

joueur frappe à nouveau la lettre Q; Cette fois, cela signifie : « Je ne désire pas ajouter d'autres questions pour le groupe 1. » Le programme passe obligeamment à la saisie des données relatives au groupe numéro 2. Le dialogue pour ce dernier groupe est analogue au précédent.

Il y a un détail de l'affichage que la figure 7.2 ne montre pas parce qu'il est difficile à rendre sur papier. A trois endroits du dialogue, la lettre Q a été utilisée. De façon à réduire les risques d'erreur, le programme, en réalité, efface la lettre Q lorsque la réponse ne se compose que de ce caractère suivi de RETURN. En conséquence, l'écran est conforme à la figure 7.2, à ceci près, que les lettres Q qui servent de signal de fin, y sont remplacées par des blancs.

Voyons maintenant, comment les commandes N, 1 et 2 peuvent être utilisées pour modifier la série de questions de la figure 7.2. Les commandes S et L n'ayant pas été programmées sur la version de Jumelage présentée dans ce livre, nous n'en parlerons pas davantage.

Commande	Effet
I	*Initialiser* : Le jeu part de zéro. Le programme demande les noms des deux groupes, puis saisit une série de questions pour chacun des groupes.
L	*Chargement* : Cette commande n'est pas réalisée dans la version du programme présentée dans ce livre. Un nom de groupe et la série de questions pour ce groupe sont chargés à partir d'une cassette ou d'un disque.
N	*Edition-Mise à jour des Noms* : Les noms des groupes sont affichés et les noms en remplacement sont acceptés.
Q	*Quitter* : Le signal " [] " est affiché et la phase de jeu commence.
S	*Sauvegarder* : Cette commande n'est pas en vigueur dans la version du programme présentée dans ce livre.
1	*Edition Groupe* 1 : Le signal " * " est affiché, et les commandes d'édition-mise à jour pour les questions du groupe 1 sont acceptées.
2	*Edition Groupe* 2 : Le signal " * " est affiché, et les commandes d'édition-mise à jour pour les questions du groupe 2 sont acceptées.

Ce sont les commandes que l'on peut entrer en réponse au signal " : : ". Les commandes I, L, et S déclenchent des processus précis. Les commandes Q, 1, et 2 appellent des routines qui permettent de choisir des commandes plus détaillées.

Figure 7.1 : Possibilités de Réponses au Signal « : : »

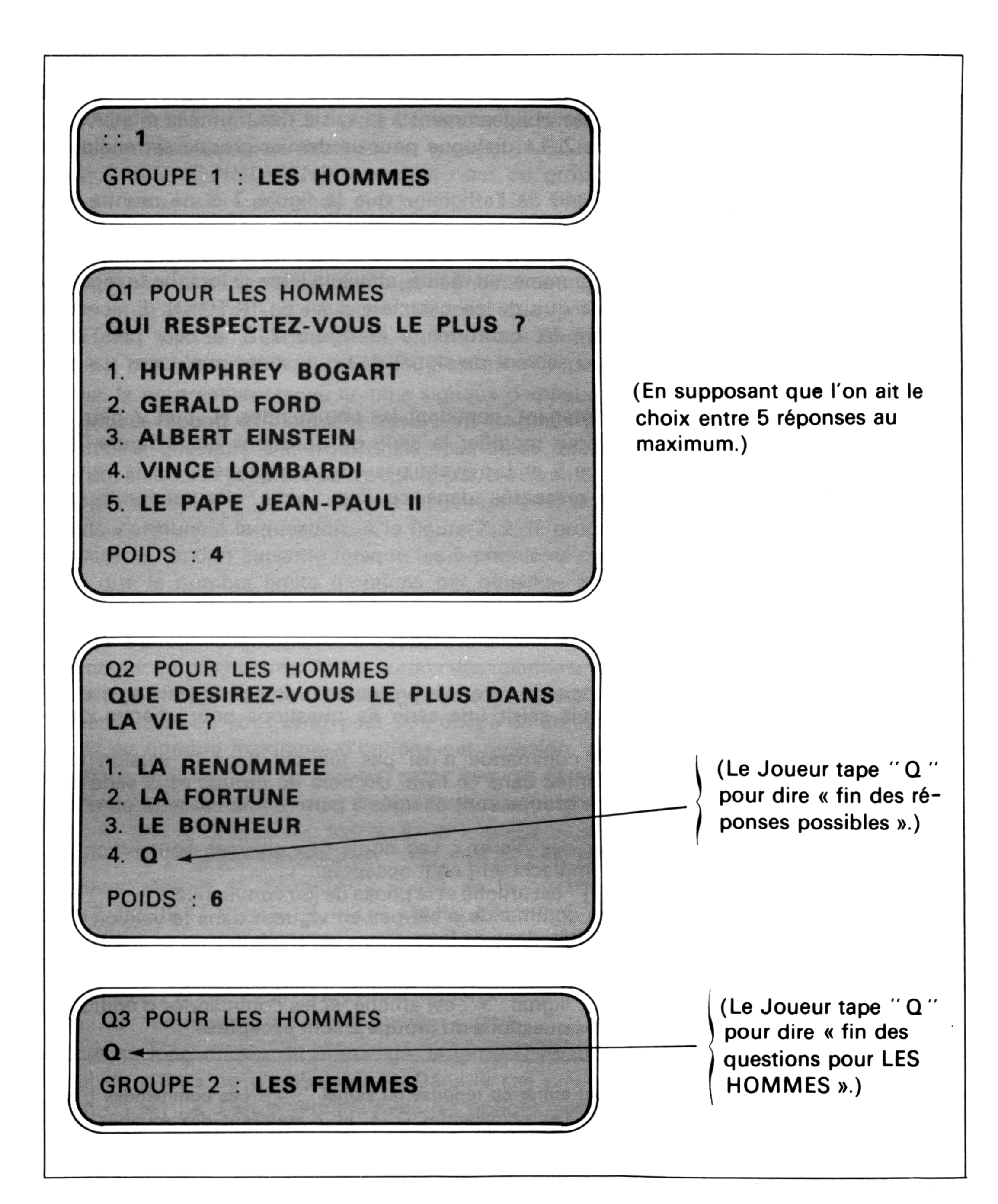


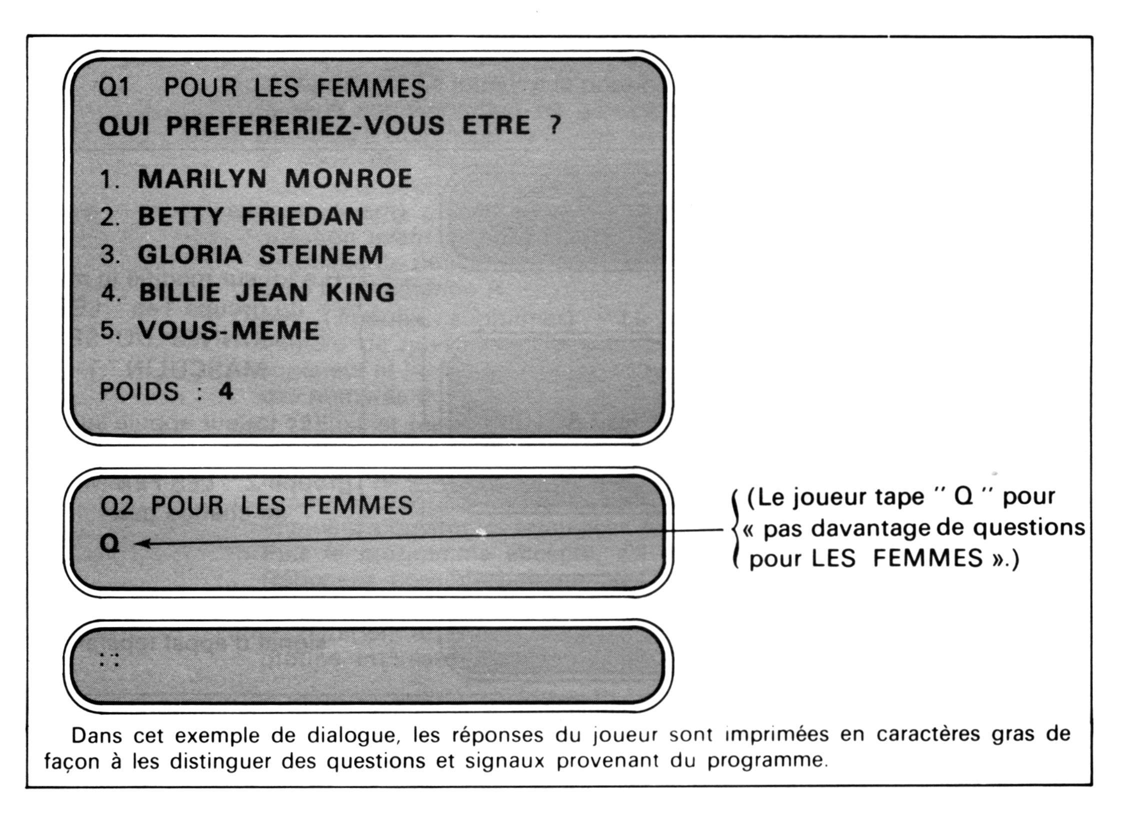
Figure 7.2 : Dialogue d'Initialisation

La commande N

La figure 7.3 donne un exemple du dialogue qui suit l'entrée de N. Les noms des deux groupes sont affichés successivement. Après l'affichage du nom, le joueur peut taper un nom de remplacement ou bien appuyer sur RETURN, auquel cas l'ancien nom sera conservé. Ici, le nom du groupe 1 " LES HOMMES " devient " LES PERSONNES DE SEXE MASCULIN " tandis que celui du groupe 2 est maintenu.

Les commandes 1 et 2

Elles sont similaires. Dans les deux cas, l'écran est effacé, le nouveau signal " * " apparaît et le programme attend l'entrée de l'une des commandes « mono-caractères » présentées à la figure 7.4.



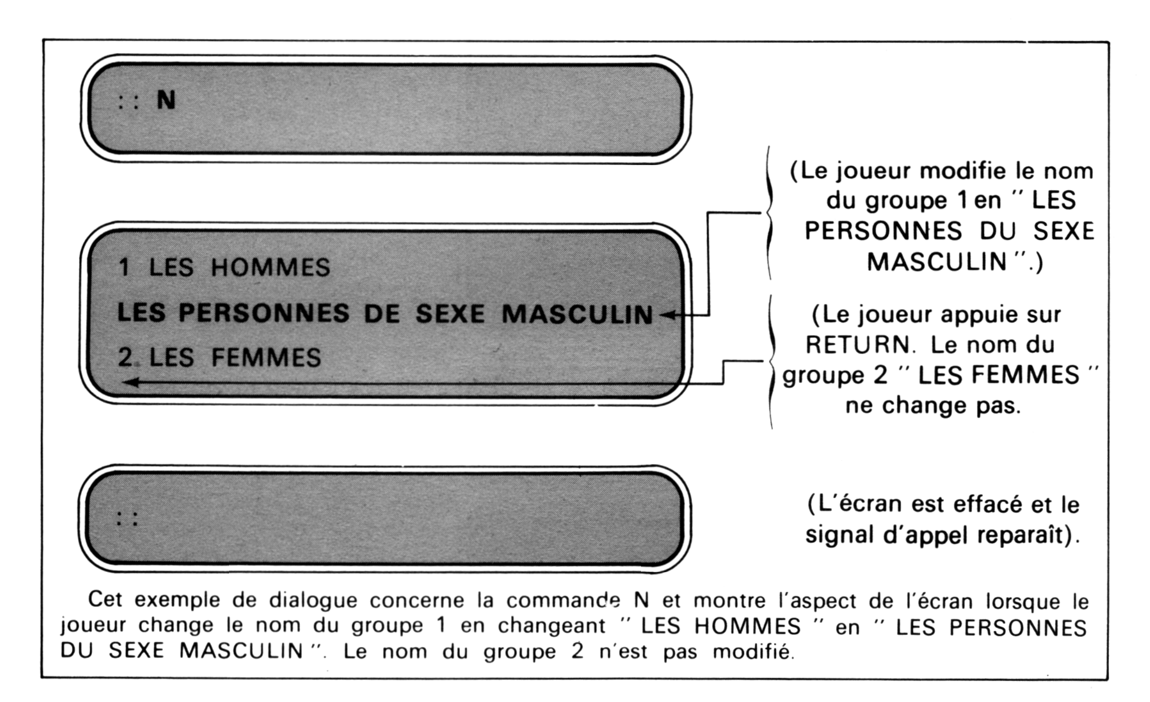
Dans cet exemple de dialogue, les réponses du joueur sont imprimées en caractères gras de façon à les distinguer des questions et signaux provenant du programme.

Figure 7.2a : Dialogue d'Initialisation

Ces dernières commandes se rapportent au groupe spécifié (groupe 1 ou groupe 2) selon que la commande 1 ou la commande 2 a été utilisée pour demander la mise à jour du fichier des questions.

La commande E

La figure 7.5 donne un exemple d'utilisation de la commande E. Dans celui-ci, le joueur met à jour la question entrée pour le groupe 2 à la figure 7.2. Le libellé de la question n'est pas modifié. La première option est modifiée, la deuxième reste la même et la troisième est supprimée. En conséquence, les réponses qui portaient auparavant les numéros quatre et cinq, deviennent maintenant les options trois et quatre. La nouvelle option 3 (qui s'appelait 4 auparavant) est modifiée et l'option qui porte le numéro 4 (qui portait le numéro 5 antérieurement) n'est, par contre, pas changée. Etant donné qu'il y a maintenant 4 options seulement, le programme signale qu'on peut entrer une cinquième réponse; mais le joueur tapant RETURN il n'y aura pas de cinquième option. Le programme



Cet exemple de dialogue concerne la commande N et montre l'aspect de l'écran lorsque le joueur change le nom du groupe 1 en changeant " LES HOMMES " en " LES PERSONNES DU SEXE MASCULIN ". Le nom du groupe 2 n'est pas modifié.

Figure 7.3 : Dialogue de Mise à Jour des Noms de Groupes

affiche alors, le coefficient en vigueur pour la question, et le joueur en modifie la valeur.

Les commandes A et B

Les commandes A et B servent à ajouter des questions à celles déjà attribuées à un groupe donné. La commande A permet d'ajouter une question à la fin de la série (fig. 7.6). Le programme, qui connaît le nombre des questions existant dans la série, affiche le nombre immédiatement supérieur à celui de la dernière question. La commande B est utilisée pour insérer une réponse dans la suite de réponses en cours (fig. 7.7). Le programme ne sait pas pour autant, où la nouvelle question doit être insérée. Aussi, commence-t-il par

Commande	**Effet**
A	*Add* (Ajouter) : Le joueur a la possibilité d'ajouter une question en fin de la série correspondant au groupe donné. Le programme appelle les données à fournir par le joueur en lui signalant le numéro de question et ceux des réponses entre lesquelles on pourra choisir, puis il réclame le coefficient (poids numérique).
B	*Before* (Avant) : Le programme demande au joueur le numéro de la question avant laquelle il faut insérer la nouvelle question. Les renseignements concernant cette dernière sont alors demandés comme dans la commande A.
D	*Delete* (Annuler, supprimer) : Le programme demande quel est le numéro de question et la supprime, décalant les questions de rang supérieur et les renumérotant afin qu'il n'y ait aucun vide dans la suite des numéros de questions.
E	*Edit* (Editer et mettre à jour) : Le programme demande un numéro de question puis affiche le libellé en cours correspondant. Le joueur peut modifier ou laisser en l'état le texte de cette question. Ensuite le programme progresse dans les différentes réponses possibles (choix), le joueur pouvant les remplacer, les supprimer, ou les laisser tels quels. Puis le programme accepte, s'il y a lieu, un supplément de choix (réponses possibles) (dans la limite de maximum autorisé). Enfin, le poids est affiché ; le joueur peut le modifier, ou ne pas le changer.
Q	*Quit* (Quitter, abandonner) : Cela termine l'édition-mise à jour du groupe intéressé. L'écran est effacé et le signal " : : " est affiché.

Voici les commandes dont on dispose lorsqu'en réponse au signal " * " on a entré soit la commande 1 soit la commande 2.

Figure 7.4 : Possibilités de Réponses au Signal "*"

```
:: 2
```

*E NUMERO DE QUESTION : **1** QUI SOUHAITERIEZ-VOUS ETRE ?	(Le joueur appuie sur RETURN.)
1. MARILYN MONROE **BO DEREK**	(Le joueur modifie la première réponse possible (choix).)
2. BETTY FRIEDAN	(Le joueur appuie sur RETURN.)
3. GLORIA STEINÉM **D**	(Le joueur annule le troisième choix.)
3. BILLIE JEAN KING **ELIZABETH HOLTZMAN**	(Le joueur modifie le nouveau troisième choix.)
4. VOUS-MEME	(Le joueur appuie sur RETURN.)
5.	(Le joueur appuie sur RETURN.)
POIDS : 4 : **7**	(Le joueur modifie le poids numérique.)

*E NUMERO DE QUESTION : 1 QUI SOUHAITERIEZ-VOUS ETRE ? 1. BO DEREK 2. BETTY FRIEDAN 3. ELIZABETH HOLTZMAN 4. VOUS-MEME 5. POIDS : 7 :	(Le joueur contrôle l'exactitude de la question modifiée en faisant appel à la fonction d'édition et en appuyant sur RETURN chaque fois que c'est possible.

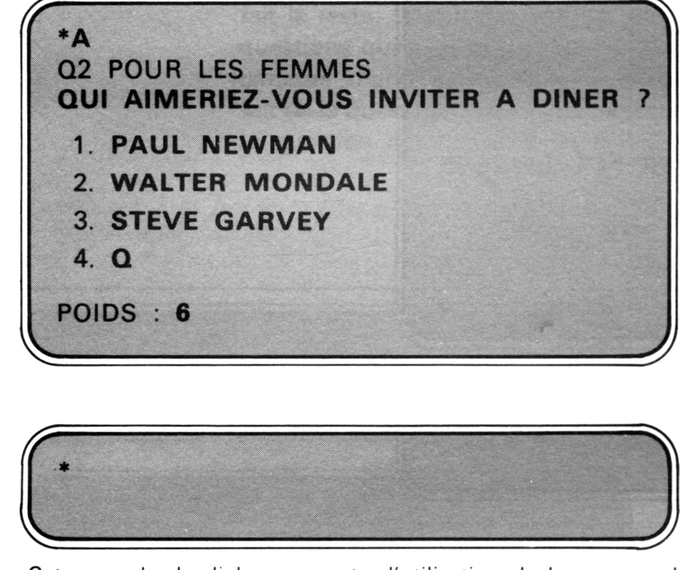
```
*
```

Cet exemple de dialogue montre comment la première question entrée pour le groupe 2 (fig. 7.2) pourrait être éditée et mise à jour. Les réponses du joueur sont indiquées en caractères plus gras. (A l'exception des réponses données directement aux signaux "::" et "*", toutes les réponses du joueur doivent être conclues par RETURN.)

Figure 7.5 : Mise à Jour des Questions

demander au joueur d'indiquer le numéro de la question avant laquelle l'insertion devra se faire. Par exemple, si le joueur précise que l'insertion est à faire avant la question numéro deux, la nouvelle question prendra le numéro deux, la question deux deviendra trois, trois deviendra quatre et ainsi de suite. Que ce soit à partir de la commande A ou de la commande B, une fois le numéro de question déterminé, les nouvelles données sont entrées exactement de la même façon que pour la commande I (fig. 7.2).

L'utilisation de la lettre B (de préférence à la lettre I) comme commande d'insertion de question, mérite d'être expliquée. En premier lieu, le joueur peut faire l'erreur (facile à commettre) d'entrer la commande « B » à partir du signal « : : » sans passer par l'intermédiaire de la commande 1 ou de la commande 2. En utilisant B, le seul inconvénient sera le rejet de la commande (puisque " B "



Cet exemple de dialogue montre l'utilisation de la commande A pour ajouter une question à la série entrée à la figure 7.2 et modifiée en figure 7.5. Le signal " * " en haut de la présente figure pourrait être en fait celui qui est affiché dans la partie inférieure de la figure 7.5.

Le message " Q2 POUR LES FEMMES " est affiché par le programme, parce qu'il s'agit des questions du groupe 2, et qu'une commande « A » entraîne l'Addition de la nouvelle question en fin de série. Si le joueur désirait mettre cette question à un endroit autre qu'en fin de série, il lui faudrait utiliser la commande « B ».

Figure 7.6 : Ajouter une Question



Cet exemple de dialogue montre l'utilisation de la commande " B ". On suppose que le joueur vient d'éditer (mettre à jour) les questions du groupe 2 ; par exemple le " * " du début pourrait être le même que celui du bas de la figure 7.6. Le joueur utilise la commande « Q » pour retourner sous signal " : : ", et la commande 1 pour déclencher le signal " * " correspondant à l'édition du groupe 1.

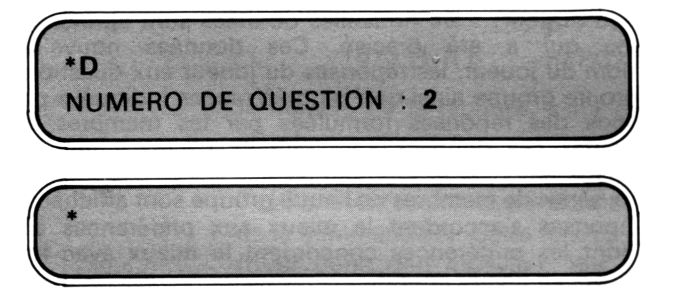
Si le fichier des questions du groupe 1 est le même que celui qui a été entré à la figure 7.2, les questions qui portaient les numéros un et deux vont devenir deux et trois, et la question nouvelle va devenir la question numéro 1.

Figure 7.7 : Insertion d'une Question

n'est pas une des commandes admises à partir du signal " : : "). L'utilisation de « I », au contraire, déclencherait malencontreusement l'initialisation et donc, forcerait à repartir de zéro. La deuxième raison qui plaide en faveur de l'utilisation de B (au lieu de I), est que la qualité mnémonique de B (pour « before », avant) aide le joueur à se rappeler que la nouvelle question sera insérée avant celle dont le numéro est précisé dans la commande. Il peut arriver que vous ne sachiez plus très bien si l'insertion doit se faire avant ou après la question indiquée comme repère, et ceci surtout si vous utilisez par ailleurs un système de mise à jour qui applique des règles différentes.

La commande D

La dernière commande servant à tenir à jour le fichier des questions, est la commande D (pour delete, supprimer) (voir figure 7.8). Le joueur indique le numéro de la question à supprimer, et le programme fait le reste. La question ne fait plus partie de la série et toutes les questions qui avaient un numéro supérieur au sien, voient leur rang diminué d'une unité. Ainsi la suppression de la question quatre fait de la question cinq, la nouvelle question quatre, de la question six, la question cinq et ainsi de suite.



Ceci donne un exemple du dialogue qui doit avoir lieu pour supprimer une question déterminée dans la série. Si l'interaction ci-dessus a lieu immédiatement après celle de la figure 7.7, la question " QUI RESPECTEZ-VOUS LE PLUS ? " entrée comme question numéro 1 à la figure 7.2 et devenue numéro 2 à la suite de l'insertion montrée à la figure 7.7, sera alors éliminée des questions de groupe 1. La question " QUE DÉSIREZ-VOUS LE PLUS DANS LA VIE ?" portera de nouveau le numéro 1.

Figure 7.8 : Suppression d'une Question

Ceci achève notre étude du jeu des commandes utilisées pour tenir à jour les fichiers de questions du groupe 1 et du groupe 2. Ces commandes assurent d'une façon complète, les fonctions de tenue à jour (en ce sens que toutes les fonctions nécessaires sont prévues), mais plusieurs possibilités très intéressantes n'existent pas. Quelques-unes d'entre elles font partie des améliorations suggérées en fin de chapitre.

La phase de jeu

La phase suivante est la phase de jeu effectif qui commence au moment où l'on utilise la commande Q (quit, quitter) en réponse à un signal " : : ". Dans cette phase de jeu, le programme se signale

Commande	Effet
E	*Edit* (Mise à jour) : Les données concernant le joueur précisé sont affichées et le joueur peut y apporter des modifications ainsi que procéder à des additions et suppressions.
L	*Load* (Chargement) : Cette commande n'a pas été réalisée dans la version du programme présentée dans ce livre Cette commande aurait pour effet de charger en mémoire les données concernant les joueurs de l'un des groupes à partir d'une cassette ou d'un disque.
N	*New Player* (Nouveau Joueur) : De nouvelles données sont ajoutées à celles du groupe qui a été précisé. Ces données nouvelles comprennent le nom du joueur, les réponses du joueur aux questions concernant son propre groupe ainsi que les préférences indiquées par ce joueur à propos des réponses formulées par les membres de l'autre groupe. Toutes ces données sont entrées par le joueur en réponse à des signaux émis par le programme.
P	*Pair* (Jumeler) : Deux séries de membres de l'autre groupe sont affichées : ceux dont les réponses s'accordent le mieux aux préférences du joueur et ceux dont les préférences concordent le mieux avec les réponses données par ce même joueur.
Q	*Quit* (Quitter, abandonner) : Partie terminée. On revient sous le signal " : : " pour commencer une nouvelle partie.
S	*Save* ("Sauvegarder") : Cette commande n'est pas réalisée dans la version du programme présentée dans ce livre. Les données concernant les joueurs de l'un des groupes sont « sauvegardées » par écriture sur cassette ou disque.

Figure 7.9 : Les Réponses Possibles au Signal "[ ]"

par " [ ] ". Il acceptera en entrée des commandes monocaractères qui servent à préciser et à modifier les données relatives à chaque joueur. C'est aussi lors de cette phase que le programme calcule le degré de compatibilité des différentes paires de joueurs. La figure 7.9 présente les commandes qu'on peut utiliser quand on est sous signal "[ ]".

La commande N

Les figures 7.10 et 7.11 présentent un exemple de dialogue possible après recours à la commande N. Le programme commence par afficher les noms des deux groupes, et par demander au joueur de bien vouloir préciser l'un d'entre eux, en entrant 1 ou 2. Ensuite il demande le nom du joueur. Une fois le nom enregistré, le programme présente au joueur les questions relevant de son groupe tel que celui-ci a été indiqué. Ledit joueur est invité à choisir l'une des réponses prévues pour chaque question. Lorsqu'il a donné son nom et fait ses choix (fig. 7.10), le joueur peut indiquer des préférences qui seront utilisées pour évaluer le degré de compatibilité entre les membres de l'autre groupe et ce qu'il souhaite lui-même. Dans l'exemple de la figure 7.11, le joueur peut associer une des réponses possibles à chacune des valeurs 2, 1, − 1 et − 2. Le nombre de ces valeurs et les valeurs elles-mêmes sont définis dans des instructions DATA qui se trouvent dans la routine d'initialisation, de sorte que cet élément du jeu peut aisément être modifié. Le programme demande sous la forme

VALEUR n REPONSE :

le numéro de la réponse, s'il y en a une à laquelle le joueur désire affecter la valeur n.

Pour illustrer la méthode d'évaluation utilisée, les figures 7.12 et 7.13 présentent un autre exemple du dialogue consécutif à l'entrée de la commande N. Cette fois-ci, le joueur appartient à l'autre groupe (c'est donc une joueuse). Ses réponses (fig. 7.12) sont rapprochées des préférences exprimées par le premier joueur (fig. 7.11). Les figures montrent bien la méthode utilisée. Pour évaluer les réponses du joueur B d'après les préférences du joueur A, une note est calculée pour chacune des réponses données par B et ces notes sont cumulées pour obtenir l'évaluation finale. La note attribuée à une question donnée est obtenue en multipliant la valeur que A a affectée au numéro de la réponse effectivement choisie par B, par le coefficient attribué à la question pendant la phase préparatoire.

```
[]N

  1. LES HOMMES
  2. LES FEMMES
GROUPE : 1
```

(Le joueur appartient au groupe des HOMMES.)

```
NOM : JOHN
```

(Le nom du joueur est JOHN)

```
QUEL AGE AVEZ-VOUS ?
  1. MOINS DE 25
  2. 25-38
  3. 39
  4. TROP VIEUX POUR,M'Y LAISSER
     PRENDRE
  5. A L'AGE DANGEREUX
CHOIX : 2
```

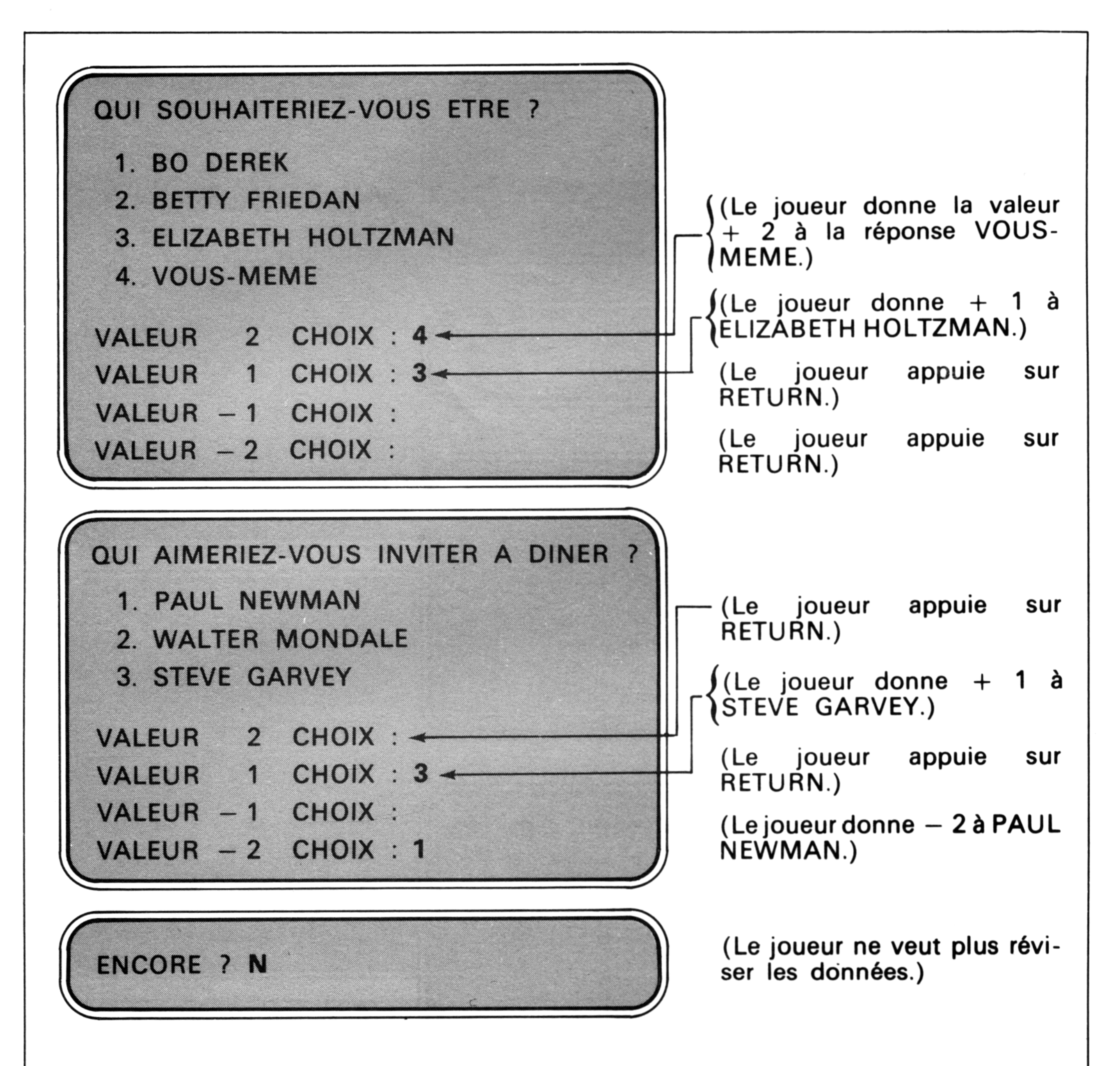
(L'âge du joueur est 25-28.)

```
QUE DESIREZ-VOUS LE PLUS DANS
LA VIE ?
  1. LA RENOMMEE
  2. LA FORTUNE
  3. LE BONHEUR
CHOIX : 3
```

(Le joueur désire LE BONHEUR.)

Cet exemple montre le début du dialogue de saisie des données concernant un nouveau joueur. Les noms des deux groupes sont d'abord affichés, et le joueur indique son appartenance au groupe des HOMMES. Le programme demande alors au joueur son nom et le joueur répond JOHN. Ensuite le programme dévide les questions du groupe 1 et JOHN choisit pour chacune une réponse. C'est à partir de là que le programme passera à la partie où JOHN sera prié d'indiquer comment, selon lui, un élément à son goût du groupe nommé LES FEMMES, devrait répondre aux questions du groupe 2. Le dialogue correspondant est présenté à la figure 7.11.

Figure 7.10 : Les Réponses de John



Cet exemple montre la partie du dialogue qui poursuit la saisie des données concernant le nouveau à partir du point atteint à la figure 7.10. JOHN fait partie du groupe 1 (HOMMES). On lui montre les questions posées aux membres du groupe 2 (FEMMES). Il a la possibilité d'affecter une valeur positive ou négative aux reprises possibles (choix) pour chacune des questions. Dans l'exemple ci-dessus, quatre valeurs sont disponibles (2, 1, – 1, – 2) que JOHN peut associer à un choix déterminé, Le nombre de valeurs (4) et des valeurs elles-mêmes (2, 1, – 1, – 2) figurent dans les instructions DATA de la routine d'initialisation. Les choix non associés à des valeurs reçoivent, par défaut, une valeur de zéro. Ces valeurs servent à évaluer la compatibilité de chacun des joueurs du groupe 2 avec les aspirations de JOHN.

Figure 7.11 : Les Préférences de John

```
[ ] N
 1. LES HOMMES
 2. LES FEMMES
GROUPE : 2
```

```
NOM : SUSAN
```

```
QUI SOUHAITERIEZ-VOUS ETRE ?
 1. BO DEREK
 2. BETTY FRIEDAN
 3. ELIZABETH HOLTZMAN
 4. VOUS-MEME
CHOIX : 2
```

```
QUI AIMERIEZ-VOUS INVITER A DINER ?
 1. PAUL NEWMAN
 2. WALTER MONDALE
 3. STEVE GARVEY
CHOIX : 1
```

Ceci est la première partie de la saisie des données d'un nouveau joueur, membre du groupe 2. Voici comment les réponses données par SUSAN sont évaluées sous l'angle des préférences exprimées par JOHN :

Question 1 :	(VALEUR : 0	× (POIDS : 7) =	0
Question 2 :	(VALEUR : – 2)	× (POIDS : 6) =	– 12
TOTAL EVALUATION.			– 12

A la figure 7.13, ce sont les préférences de SUSAN qui servent à évaluer les réponses de JOHN.

Figure 7.12 : Les Réponses de Susan

```
QUEL AGE AVEZ-VOUS ?
  1. MOINS DE 25
  2. 25-38
  3. 39
  4. TROP VIEUX POUR M'Y LAISSER
     PRENDRE
  5. A L'AGE DANGEREUX
VALEUR  2   CHOIX : 2
VALEUR  1   CHOIX : 3
VALEUR -1   CHOIX : 4
VALEUR -2   CHOIX : 5
```

(Le joueur donne la valeur + 2 à la réponse **25-38**.)

(Le joueur donne **+ 1 à 39.**)

(Le joueur donne **– 1 à TROP** VIEUX ... PRENDRE.)

(Le joueur donne **– 2 à AGE** DANGEREUX.)

```
QUE DESIREZ-VOUS LE PLUS DANS
LA VIE ?
  1. LA RENOMMEE
  2. LA FORTUNE
  3. LE BONHEUR
VALEUR  2   CHOIX : 3
VALEUR  1   CHOIX :
VALEUR -1   CHOIX : 2
VALEUR -2   CHOIX : 1
```

(Le joueur donne **+ 2 au BON**-HEUR.)

(Le joueur appuie sur **RETURN**.)

(Le joueur donne **– 1 à LA** FORTUNE.)

(Le joueur donne **– 2 à LA** RENOMMEE.)

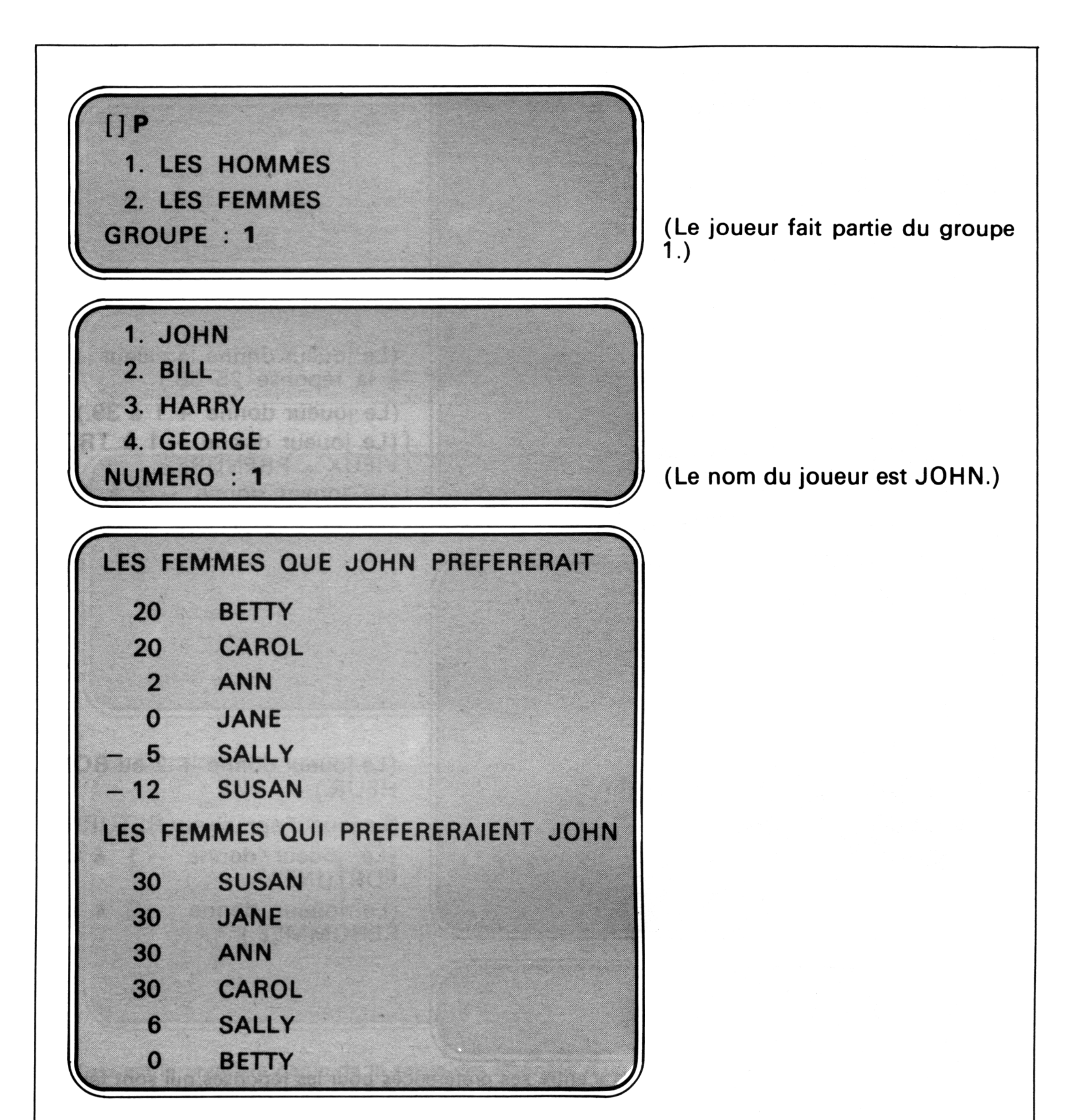
```
ENCORE ? N
```

Dans cet exemple de dialogue, SUSAN a entré ses préférences pour les réponses qui sont faites par les membres du groupe 1 (LES HOMMES). Voici comment les réponses faites par JOHN sont évaluées d'après les préférences exprimées par SUSAN :

Question 1 : (VALEUR : 2) × (POIDS : 9) = 18
Question 2 : (VALEUR : 2) × (POIDS : 6) = 12

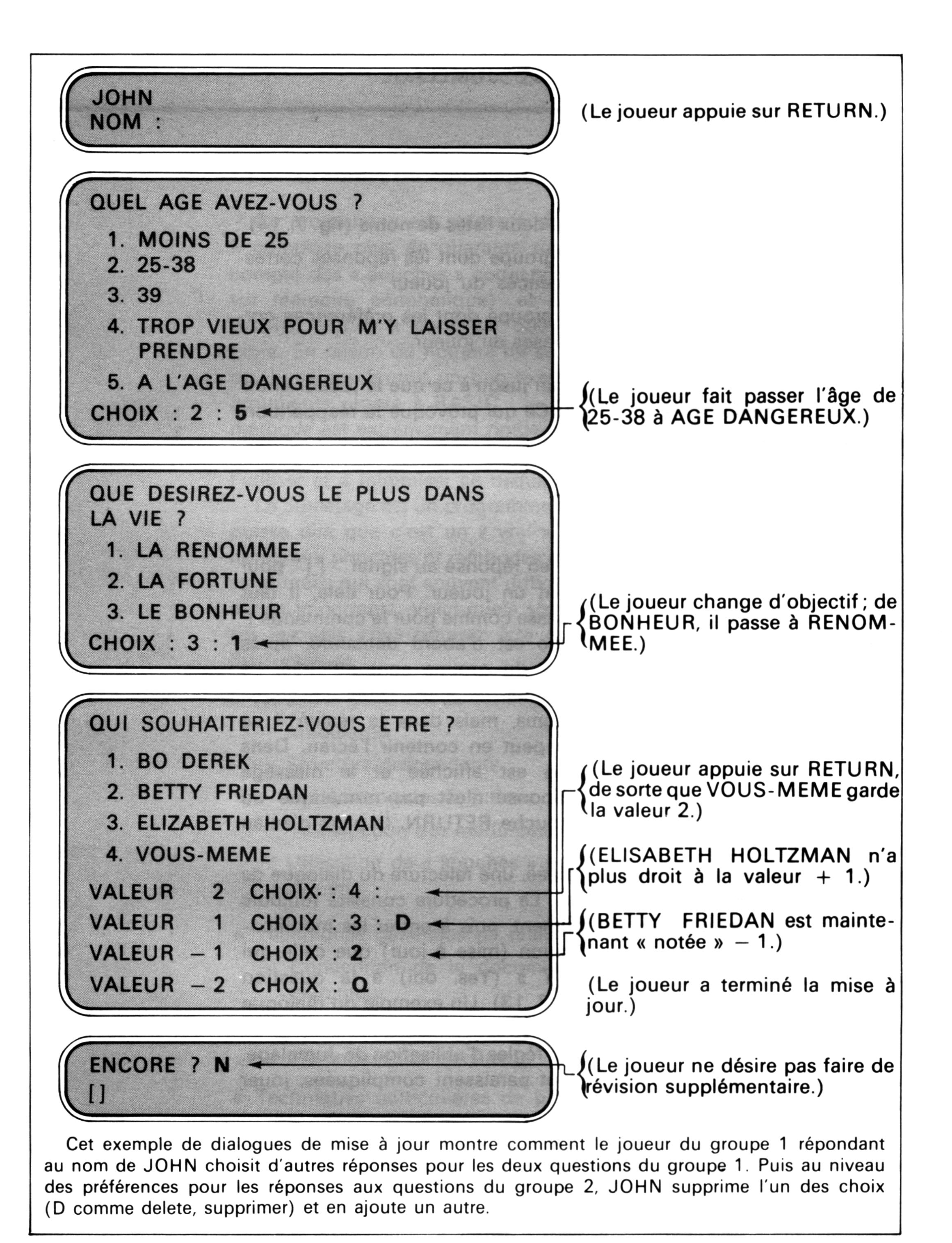
TOTAL EVALUATION : 30

Figure 7.13 : Les Préférences de Susan



Cet exemple montre l'utilisation de la commande « P » (comme « Pairing ») pour obtenir deux listes : celle des joueurs de l'autre groupe dont les réponses se rapprochent le plus de ce que souhaite JOHN et celle des joueurs de ce même autre groupe dont les préférences cadrent le mieux avec les réponses données par JOHN. Le nombre qui figure à gauche de chaque nom est l'évaluation exacte.

Figure 7.14 : Les Couples



Cet exemple de dialogues de mise à jour montre comment le joueur du groupe 1 répondant au nom de JOHN choisit d'autres réponses pour les deux questions du groupe 1. Puis au niveau des préférences pour les réponses aux questions du groupe 2, JOHN supprime l'un des choix (D comme delete, supprimer) et en ajoute un autre.

Figure 7.15 : Mise à Jour des Données concernant un Joueur

La commande P

La commande P sert à afficher les deux listes de noms (fig. 7.14) :

- celle des membres de l'autre groupe dont les réponses correspondent le mieux aux préférences du joueur,
- celle des membres de l'autre groupe dont les préférences correspondent le mieux aux réponses du joueur.

Ces renseignements restent sur l'écran jusqu'à ce que le joueur entre un caractère quelconque au clavier. Ce qui provoque la réapparition du signal « [ ] ».

La commande E

La commande E peut être utilisée en réponse au signal " [ ] " pour modifier les informations concernant un joueur. Pour cela, il faut d'abord identifier le joueur. Cela se passe comme pour la commande P (fig. 7.14) : le numéro du groupe est d'abord demandé, après quoi, les noms de tous les joueurs du groupe sont affichés. Le numéro du joueur concerné est alors réclamé. Dans l'exemple présenté, il n'y a que quelques noms, mais dans la réalité il se pourrait qu'il y en ait plus que ne peut en contenir l'écran. Dans ce dernier cas, une liste partielle est affichée et le message " NUMERO : " apparaît. Si la réponse n'est pas numérique ou correspond à l'enfoncement de la touche RETURN, la page d'écran suivante est affichée.

Une fois l'identité du joueur précisée, une relecture du dialogue de saisie des données du joueur a lieu. La procédure consiste toujours à lister d'abord les données du moment, puis à entrer les modifications. C'est la même fonction d'édition (mise à jour) que celle qui serait appelée par la réponse « Y » (Yes, oui) à la question " ENCORE ? " (voir figures 7.11 et 7.13). Un exemple du dialogue de mise à jour est donné en figure 7.15.

Ceci termine notre présentation des règles d'utilisation de Jumelage. Alors que ces règles sont longues et paraissent compliquées, jouer à Jumelage est en fait, très simple.

---

* La chaîne monocaractère « Y » n'a pas été traduite en « O » (pour Oui) car la lettre O est souvent confondue avec le 0 (zéro) dont elle est d'ailleurs très proche sur le clavier. *N. du T.*

Le programme du Jumelage

Ce programme est présenté de la figure 7.16 à la figure 7.59. Il comporte plus de quarante routines (même si l'on ne tient pas compte des « souches » correspondant à des fonctions de stockage sur mémoire périphérique), et plus de 225 lignes d'instructions BASIC, écrites à partir de 500 lignes de description en BASIC Libre. En raison du nombre de ses sous-programmes, le programme du Jumelage fournit un bon exemple du mode de transmission des arguments propre à BASIC : le passage global « implicite ». Cette méthode est extrêmement rigide et propice aux erreurs, mais si vous vous apprêtez à programmer en BASIC, vous devrez apprendre à l'utiliser et à minimiser ce risque d'erreurs.

Le Jumelage est un programme suffisamment important pour qu'on puisse dire que c'est un « vrai » programme ; il met en lumière de nombreux principes et méthodes de conception et de programmation structurées qui sont souvent difficiles à rencontrer dans des exemples moins importants. Voici quels sont les aspects de la programmation sur lesquels vous pourrez être éclairés par l'étude du programme du Jumelage :

- Principes généraux de conception :
  - — Modularité
  - — Structure descendante
  - — Localisation des fonctions
  - — Anticipation des modifications
  - — Utilisation de « souches » pour construire la structure du programme.

- Techniques générales de réalisation des programmes :
  - — Conception de la documentation de l'utilisateur et du programmeur
  - — Conception des interactions homme-machine
  - — Traitement des erreurs.

- Techniques particulières de programmation :
  - — Conception des fonctions de mise à jour
  - — Algorithmes d'insertion et de suppression
  - — Conception des fonctions de gestion de fichiers
  - — Economie de la place mémoire.

Nous allons voir maintenant ces points plus en détail.

Principes généraux de conception

C'est la première catégorie de la liste ci-dessus. Sous ce titre, nous allons débattre de certains points de développement des programmes qui ont une portée générale.

La modularité. L'expression « modularité » n'a pas de définition universellement reconnue. Ici, nous l'utilisons pour attirer l'attention sur le fait que le programme du Jumelage a été découpé en éléments séparés (comme c'est d'ailleurs mis en évidence par sa description sous forme de quarante figures autonomes). Chacun de ces éléments a une fonction facilement identifiable à remplir et des procédures précises d'interaction avec les autres éléments. L'application mise à définir parfaitement les interfaces avec les autres éléments du programme, est particulièrement importante en BASIC où la totalité du programme peut accéder à la totalité des variables.

La structure descendante. La structure descendante est illustrée par le programme du Jumelage. Le programme « principal » vient en tête; puis les routines qu'il appelle, enfin les routines appelées par celles qui constituent le programme principal et ainsi de suite. Bien sûr, ce que nous venons de décrire est une structure de développement à deux dimensions (c'est un « arbre ») et ce livre en raison de la succession de ses pages en une structure linéaire, n'en donne qu'une représentation approchée.

Cette façon de structurer les programmes aide le lecteur en lui facilitant la compréhension de ceux-ci. Chaque « niveau » du programme est constitué d'appels à des éléments de programme de niveau inférieur dont le rôle, au fur et à mesure, devient plus spécialisé. Le degré d'abstraction diminue à chaque niveau : au niveau le plus élevé, le programme traite des concepts abstraits tels que : ajouter des joueurs ou chercher des paires de personnes compatibles; au niveau le plus bas, des instructions BASIC précises et détaillées sont utilisées pour traiter des variables spécifiques.

Néanmoins, le qualificatif « descendant » va d'habitude au-delà du listage du programme. Ce terme s'applique plutôt à l'ordre dans lequel se fait le développement des différentes parties du programme. Effectivement, Jumelage a été développé en suivant presque exactement le même ordre que celui dans lequel ses diverses parties sont présentées dans ce livre.

Nous avons donné de nombreux exemples de conception et de réalisation de programmes. Cela ne peut suffire cependant, pour traiter l'ensemble de la question. Pour étudier la programmation

```
# Jeu du Jumelage
GOSUB init                                      # préciser tableaux, constantes
repeat {
GOSUB creation                                  # créer ou mettre à jour le fichier des questions
repeat {                                        # jouer la partie
      GOSUB effacecran
      PRINT " [ ] ";                            # afficher signal
      GOSUB uncar : PRINT X$                    # saisir et renvoyer/écran la commande
      IF case
           X$ = " E " THEN GOSUB editjoueur     # édition/mise à jour données concernant un joueur
           X$ = " N " THEN GOSUB nouveaujoueur  # ajouter un nouveau joueur
           X$ = " P " THEN GOSUB jumelage       # afficher les couples possibles pour un joueur
           X$ = " L " THEN GOSUB chargerjoueurs # charger le fichier des données « joueurs »
           X$ = " S " THEN GOSUB sauverjoueurs  # sauvegarder le fichier « joueurs »
           X$ = " Q " THEN {                    # fin de la partie
                PRINT " VRAIMENT ? "
                GOSUB uncar
                IF X$ = " Y " THEN
                     break
                }
      }
}
100 GOSUB2260
110 GOSUB1420
120 GOSUB2200:PRINT"[ ]";:GOSUB2180:PRINTX$
130 IFX$="E"THENGOSUB210:GOTO120
140 IFX$="N"THENGOSUB260:GOTO120
150 IFX$="P"THENGOSUB970:GOTO120
160 IFX$="L"THENGOSUB2230:GOTO120
170 IFX$="S"THENGOSUB2240:GOTO120
180 IFX$<>"Q"THEN120
190 PRINT"VRAIMENT?";:GOSUB2180
200 IFX$="Y"THEN110
205 GOTO120
```

Voici la boucle maîtresse du jeu du Jumelage. La routine « création » contient un menu analogue permettant de choisir une des commandes adaptées à la construction du fichier des questions. La boucle intérieure présentée ici met en œuvre des commandes pour « jouer » la partie, construire le fichier « joueurs » et demander l'affichage des couples ayant des affinités.

Figure 7.16 : Le Jumelage

" E " — Commande d'édition des données « joueurs » préexistantes

```
editjoueur  GOSUB identite                    # demander (PP, GP)
            If XX = 0 THEN                    # sortie en cas d'erreur
                 RETURN
            GOSUB autregroupe                 # GC = groupe « autre » que
                                              GP
            GOSUB chargerzone                 # charger données dans zone
                                              travail
            repeat {
                 GOSUB modifs                 # saisie des modifs
                 GOSUB çava                   # demander si encore modifs
                 } until (OK = 1 OR OK = — 1) # mise à jour ou abandon
            IF OK = 1 THEN
                 GOSUB misajour               # mise à jour des données à par-
                                              tir zone travail
            RETURN
```

```
210  GOSUB1310:IFXX=0THENRETURN
220  GOSUB1290:GOSUB340
230  GOSUB460:GOSUB510:IFNOT(OK=1OROK=-1)THEN230
240  IFOK=1THENGOSUB380
250  RETURN
```

La routine « editjoueur » est la routine « d'édition » des données d'un joueur. Le sous-programme « identité » saisit un indice PP correspondant au joueur et un autre GP, à son groupe, et ceci au cours d'un dialogue avec le joueur. La routine « autregroupe » calcule l'indice GC du groupe qui doit être associé au groupe du joueur.

La routine « chargerzone » va chercher les éléments de tableau correspondant au joueur (PP, GP), et les place dans une zone de travail où on peut les manipuler sans modifier les éléments réels du tableau. La routine « modifs » passe en revue les informations qui se trouvent dans la zone de travail et permet leur modification. La routine « çava » est alors appelée. La variable OK y est utilisée de la façon suivante : « modifs OK, mettre à jour » (OK = 1), « modifs encore à faire » (OK = 0), ou « abandonner le processus de mise à jour » (OK = – 1). En dernier lieu la routine « misajour » renvoie le contenu (après « edition ») de la zone de travail dans les éléments de la table d'où l'information venait.

Si le processus est abandonné à quelque moment que ce soit avant l'appel de « misajour », les données antérieures ne seront pas modifiées.

Figure 7.17 : Edition et Mise à Jour des Données concernant un Joueur

structurée et les avantages que celle-ci présente par rapport aux autres méthodes de développement, vous pouvez consulter les nombreux et remarquables ouvrages que Dijkstra, Wirth, Yourdon et bien d'autres, ont écrits sur les techniques de conception et de réalisation des programmes. *How To Manage Structured Programming* d'Ed Yourdon contient une présentation particulièrement claire des études en question.

Localisation des fonctions. Non seulement il importe que les programmes soient segmentés en parties de faible taille, possibles à dominer, remplissant des fonctions et possédant des interfaces

```
# Créer un nouveau joueur
nouveaujoueur  GOSUB demander groupe            # obtenir le groupe (GP) du
                                                nouveau joueur
               GOSUB autregroupe                # GC est l'autre groupe
               GOSUB effacerzone                # initialisation de la zone de
                                                travail
               repeat {                         recueillir de nouvelles don-
                                                nées, puis les reviser
                    GOSUB modifs
                    GOSUB çava
                    } until (OK = 1 OR OK = – 1)  # jusqu'à ce que le joueur
                                                soit content ou abandonne
               IF OK = 1 THEN                   # s'il est content
                    GOSUB stockernouveau        # stocker alors les données
                                                dans le tableau
               RETURN

260 GOSUB1260:GOSUB1290:GOSUB300
270 GOSUB460:GOSUB510:IFNOT(OK=1OROK=–1)THEN270
280 IFOK=1THENGOSUB390
290 RETURN
```

La routine « nouveaujoueur » crée les données correspondant à un nouveau joueur. La routine « demandergroupe » se procure le numéro du groupe du joueur (GP), puis « autregroupe » en déduit le numéro (GC) de l'autre groupe. Puis la routine « effacerzone » initialise la zone de travail (c'est-à-dire la remplit de zéros et de chaînes nulles). La boucle qui contrôle l'appel de « modifs » et de « çava » est identique à celle qui se trouve dans « editjoueur ». Ceci parce que « modifs » et ses routines subordonnées sont conçues pour modifier des données existantes ou en créer de nouvelles. Si le joueur estime correct le contenu de la zone de travail, la routine « stockernouveau » est appelée pour affecter les indices adéquats aux données et les stocker.

Figure 7.18 : Création d'un Nouveau Joueur

précis, mais encore faut-il que les données, qu'il n'est pas nécessaire de disperser aux « quatre coins » du programme, soient confinées le plus possible au même endroit.

A titre d'exemple, les routines *compression* et *expansion* (fig. 7.32) sont les seuls endroits dans le programme où les associations « réponses-valeurs de préférence » faites par le joueur, sont utilisées sous leur forme réelle de nombre unique codé (fig. 7.11 et 7.13). A l'extérieur de ces routines, le nombre codé en question est traité comme « indéchiffrable », tandis que la donnée « attribution de réponses aux valeurs de préférence », est représentée par deux tableaux en correspondance, celui des réponses et celui des valeurs. La routine *compression* part de ce tableau pour former le nombre unique codé; la routine *expansion* part d'un nombre codé pour générer le tableau des numéros de réponses. Les fonctions de transformation qui font le va-et-vient entre ces deux formes de liaison « réponses-valeurs », sont « confinées » dans les deux routines de la figure 7.32. Ceci est un bon exemple de localisation de fonction. Nous en verrons un autre exemple dans la routine *score*

```
# Effacer zone de travail pour des données du groupe GP (l'autre est GC)

effacerzone  WN$ = ""                     # effacer le nom
             FOR QX = 1 TO NQ(GP)         # effacer les réponses aux
                                            questions GP
                 WA(QX) = 0
                 NEXT QX
             FOR QX = 1 TO NQ(GC)         # effacer les préférences
                                            concernant les réponses faites
                                            par GC
                 WW(QX) = 0
                 NEXT QX
             RETURN

300 WN$=""
310 FORQX=1TONQ(GP):WA(QX)=0:NEXTQX
320 FORQX=1TONQ(GC):WW(QX)=0:NEXTQX
330 RETURN
```

La routine « effacerzone » nettoie la zone de travail qui comprend le nom du joueur WN$, le tableau des réponses du joueur WA, et celui des préférences WW.

Figure 7.19 : Nettoyage de la Zone de Travail

qui assure la fonction d'évaluation réciproque des joueurs (voir figure 7.34).

Anticipation des modifications. L'anticipation des modifications est un autre principe général de conception illustré par le programme du Jumelage. Quand on conçoit des programmes, il est de la plus grande importance de tenir pour vrai, le corollaire suivant de la loi de Murphy :

> *Quelque immuable qu'un programme puisse paraître, il se trouvera toujours quelqu'un pour le modifier sous la pression d'une impérieuse nécessité, sans tenir compte des spécifications initiales.*

ou dit plus simplement :

> *Si rien ne peut être modifié, tout le sera immanquablement.*

Charger la zone de travail à partir des coordonnées joueurs (PP, GP) ; l'autre groupe est GC

```
chargerzone  WN$ = NM$(PP,GP)             # charger le nom
             FOR QX = 1 TO NQ(GP)         # charger les réponses aux
                                          questions du groupe GP
                  WA(QX) = A(QX,PP,GP)
                  NEXT QX
             FOR QX = 1 TO NQ(GC)         # charger préférences pour
                                          réponses du groupe GC
                  WW(QX) = W(QX,PP,GP)
                  NEXT QX
             RETURN
```

```
340 WN$=NM$(PP,GP)
350 FORQX=1TONQ(GP):WA(QX)=A(QX,PP,GP):NEXTQX
360 FORQX=1TONQ(GC):WW(QX)=W(QX,PP,GP):NEXTQX
370 RETURN
```

La routine « chargerzone » recopie en zone de travail le nom du joueur qui se trouve dans le tableau NM$, ainsi que les réponses du joueur (tableau A) et ses préférences (tableau W). Les valeurs des indices « joueur » et « groupe » sont données par PP et GP.

Figure 7.20 : Chargement de la Zone de Travail

Sans expliquer davantage pourquoi l'anticipation des modifications est souhaitable, examinons quelques-unes des démarches suivies pour le programme du Jumelage. Le premier de tous les principes qui permettent d'anticiper les modifications, est d'écrire les programmes de façon claire, structurée et documentée.

La méthode qui vient immédiatement ensuite par ordre d'importance pour faciliter les changements ultérieurs, consiste à éviter l'utilisation des nombres sous forme de constantes. Dans la plupart des programmes BASIC, la meilleure occasion d'appliquer cette règle se trouve dans la fixation des limites des variables de contrôle des boucles, et dans le dimensionnement des tableaux. Dans le programme du Jumelage, les dimensions des tableaux et les limites supérieures des boucles sont presque toujours des variables ; quand elles ne sont pas des variables, ce sont des paramètres (c'est-à-dire des constantes qui ont reçu un nom symbolique). Les limites inférieures des boucles, au contraire, sont pratiquement toutes précisées par des constantes (habituellement 1 ou 0). Une telle manière de faire pour ces limites inférieures rend les choses plus faciles à comprendre qu'avec des variables ; de plus, une limite de valeur 0 ou 1 a rarement besoin d'être modifiée.

```
# Mettre à jour les données joueurs (PP, GP) à partir de la zone de travail ; l'autre groupe
est GC

misajour  UP = PP : UG = GP : UC = GC            # arguments de « stockerzone »
          GOSUB stockerzone
          RETURN

380  UP=PP:UG=GP:UC=GC:GOSUB420:RETURN
```

« misajour » est le compagnon de route du sous-programme « chargerzone ». La séquence du processus « d'édition » est la suivante : GOSUB chargerzone, effectuer la révision dans la zone de travail, GOSUB misajour. La routine « stockerzone » (même si son nom laisse penser qu'elle est la compagne de « chargerzone »), utilise comme arguments UP, UG pour le joueur et UC pour « l'autre » groupe. La raison en est que « stockerzone » est appelée avec des arguments différents par « stockernouveau » ; si PP et GP étaient utilisés comme arguments par « stockerzone », la routine « misajour » serait alors obligée de modifier la valeur de PP. PP ayant une signification dans la routine qui appelle « misajour », ce changement de valeur de PP serait un « effet secondaire » de l'appel de « misajour ». Les effets secondaires des subroutines sont difficiles à éviter en BASIC, mais on doit tout faire pour y arriver.

Figure 7.21 : Mise à Jour des Données Joueur à partir de la Zone de Travail

Un autre procédé pour permettre les changements consiste à identifier les aspects du programme qui seront vraisemblablement appelés à changer et à s'assurer que ces modifications pourront être faites en un seul et unique endroit du programme. Par exemple dans le programme du Jumelage, le nombre maximum de réponses possibles associées à chaque question, est représenté par la variable MC. La valeur de cette variable est précisée dans le sous-programme d'initialisation (fig. 7.59) par les deux instructions

```
READ MQ, MC, MP
DATA maxquestions, maxchoix, maxjoueurs
```

(Dans le programme en BASIC Réel, ce sont des valeurs numériques et non des noms symboliques de paramètres qui apparaissent dans l'instruction DATA.) C'est à cet endroit et à cet endroit seulement, qu'il faut faire une modification si l'on veut changer le nombre maximum de réponses possibles, car dans tout le programme, MC a servi au dimensionnement des tables et à la valeur limite de boucles. Si l'on avait utilisé des constantes, il faudrait les retrouver toutes et les modifier. Qui plus est, si le programme est bourré de constantes numériques et si le nombre de réponses possibles par question venait à changer en passant, par exemple, de 3 à 5, les constantes 3 à modifier, devraient être d'abord distinguées de leurs consœurs qui ont aussi la valeur 3 mais qui, ayant un autre objet, n'ont pas à être modifiées. Il se peut aussi que vous ayez écrit

```
IF X > 2
```

à la place de

```
IF X > = 3
```

et dans ce cas, le passage de 3 à 5 du nombre maximum de réponses entre lesquelles choisir, implique que vous changiez le 2 en 4. Si l'instruction avait été

```
IF X > MC - 1
```

la modification interviendrait de façon automatique.

La routine d'initialisation de la figure 7.59 permet d'effectuer des changements d'une autre façon. Toutes les variables, dont les valeurs sont susceptibles d'être modifiées, ont des valeurs précisées dans un ordre DATA. Notez que d'une façon différente, la variable GM, qui représente le nombre de groupes de joueurs, est valorisée par l'instruction

```
GM = 2
```

Une instruction d'affectation est utilisée au lieu d'un ordre DATA et la constante « 2 » à la place d'un paramètre comme « max-groupes ». La raison en est qu'il n'est pas possible de changer le nombre de groupes, sans modifier complètement la structure du jeu. Nous avons écrit le nom de variable GM tout au long du programme, au lieu de « 2 », parce que nous voulions que le programme soit clair et compréhensible et non parce que nous pensions que la valeur de GM pouvait un jour changer.

Nous avons maintenant suffisamment souligné les précautions prises dans le programme du Jumelage, pour en permettre la modification. Au fur et à mesure que nous descendrons dans le détail du programme, nous trouverons de nombreux cas où des variables ont été utilisées dans ce but. Par contre, une constante est utilisée d'un bout à l'autre du programme : c'est la constante littérale « Q ».

```
# Stocker données nouveau joueur dans groupe GP
stockernouveau  UG = GP : UC = GC                  # utiliser GP et GC
                                                    courants
                IF NP(GP) < MP THEN {              # si place pour joueur
                                                    supplémentaire,
                     NP(GP) = NP(GP) + 1           # l'ajouter
                     UP = NP(GP)
                     GOSUB stockerzone
                     }
                else                               # sinon,
                     " DESOLE, PAS DE PLACE. "     # ne pas le faire
                RETURN
```

```
390  UG=GP:UC=GC
400  IFNP(GP)<MPTHENNP(GP)=NP(GP)+1:UP=NP(GP):GOSUB420:RETURN
410  PRINT"DESOLE, PAS DE PLACE.":RETURN
```

La routine « stockernouveau » est appelée après la création des données concernant le nouveau joueur. Elle affecte l'indice disponible suivant au nouveau joueur et appelle « stockerzone » pour sauvegarder les données joueur que l'on vient de mettre au point dans la zone de travail.

Cette routine illustre une erreur de conception courante. Le programmeur ne s'est pas rendu compte à temps qu'un test serait nécessaire pour savoir si oui ou non on dispose de la place nécessaire pour stocker les nouvelles données. (Ce test a sa vraie place au début de la routine « nouveaujoueur ».) Tel que c'est écrit, le joueur peut dérouler le processus complet de réponses aux questions et d'indication de ses préférences, pour s'entendre dire ce que le programme aurait pu savoir depuis le début : " DESOLE, PAS DE PLACE. ".

Figure 7.22 : Enregistrement d'un Nouveau Joueur *(Programme défectueux)*

Peut-être trouverez-vous intéressant l'exercice qui consiste à remplacer le caractère d'« abandon » « Q » (utilisé en de nombreux endroits du dialogue par un autre caractère). Savez-vous pourquoi une constante est utilisée et non pas une variable caractères ?

Utilisation de souches pour construire la structure du programme. Le dernier des principes généraux de conception que nous commenterons à présent, est celui de l'utilisation des souches. Il s'agit là d'un autre aspect du développement descendant, démarche qui implique qu'on définisse et qu'on vérifie les interactions des modules de niveau supérieur, avant de procéder au codage et à l'essai des modules au niveau suivant. De façon à pouvoir essayer les interactions des modules qui n'ont pas été encore complètement codés (et donc contrôlés), le programmeur remplace les parties manquantes par des *souches*, éléments de programme dont les interactions sont similaires à celles des parties manquantes des modules, mais dont les fonctions restent rudimentaires.

Par exemple, les programmes qui ont la charge du stockage des données concernant les questions ou les joueurs, sur mémoire externe, sont tout simplement des programmes factices qui annoncent

```
# Stocker contenu zone de travail dans la zone joueur (UP, UG) ; l'indice de l'autre
groupe est UC

stockerzone  NM$(UP,UG) = WN$                  # stocker le nom
             FOR QX = 1 TO NQ(UG)              # stocker les réponses
                  A(QX,UP,UG) = WA(QX)
                  NEXT QX
             FOR QX = 1 TO NQ(UC)              # stocker les préférences
                  W(QX,UP,UG) = WW(QX)
                  NEXT QX
             RETURN

420  NM$(UP,UG)=WN$
430  FORQX=1TONQ(UG):A(QX,UP,UG)=WA(QX):NEXTQX
440  FORQX=1TONQ(UC):W(QX,UP,UG)=WW(QX):NEXTQX
450  RETURN
```

La routine « stockerzone » recopie le contenu de la zone de travail dans le tableau NM$ pour le nom du joueur, le tableau A pour les réponses, et le tableau W pour les préférences. Les indices du joueur et du groupe sont donnés par UP et UG.

Figure 7.23 : Enregistrement du Contenu de la Zone de Travail

Saisie des modifications ou additions en zone de travail pour un joueur du groupe GP ; l'autre groupe est GC

```
modifs  GOSUB changernom                           # nom du joueur
        FOR QX = 1 TO NQ(GP)                       # réponses du joueur
            GOSUB changechoix
            IF X$ = "Q" THEN                       # Q = fini pour les modifs
                break
            NEXT QX
        IF X$ <> "Q" THEN
            GOSUB effacecran
            PRINT "demande preferences" ; GN$(GC)
            GOSUB uncar
            FOR QX = 1 TO NQ(GC)                   # préférences du joueur
                GOSUB changerpref
                IF X$ = "Q" THEN                   # Q = fini pour les modifs
                    break
                NEXT QX
        RETURN
```

```
460  GOSUB550
470  FORQX=1 TONQ(GP):GOSUB680:IFX$<>"Q"THENNEXTQX
480  IFX$="Q"THENRETURN
485  GOSUB2200:PRINT"SVP VOS PREFERENCES SUR REPONSES FAITES PAR";GN$
     (GC)
490  GOSUB2180:FORQX=1TONQ(GC):GOSUB880:IFX$<>"Q"THENNEXTQX
500  RETURN
```

La routine « modifs » demande au joueur les modifications ou additions à apporter aux données présentes dans la zone de travail. (Quand elle est en un premier temps appelée par « nouveaujoueur », cette routine ne peut demander que des additions, puisqu'il n'y a rien à changer dans la zone de travail.) La structure de cette routine est analogue à celles de « effacerzone », « chargerzone » et « stockerzone ». La routine « changernom » est appelée pour le nom, « changechoix » pour les réponses du joueur aux questions du groupe GP et « changepref » pour les préférences du joueur concernant les réponses du groupe GC.

Figure 7.24 : Modification des Données concernant un Joueur

« STOCKAGE EXTERNE NON REALISE » chaque fois qu'on les appelle. Des souches un peu plus « feuillues » — en principe plus utiles pour vérifier le fonctionnement du reste du système — auraient pu créer un fichier « questions » ou « joueurs » en mémoire centrale à chaque fois qu'un chargement était demandé. Pendant les essais réels, une solution différente a été, en fait, adoptée. Le programmeur a commencé par écrire des souches pour la partie « création » du programme (fig. 7.45 à 7.56). Ces souches ont permis la création, à partir du clavier, des données concernant les questions et le joueur avec des fonctions de mise à jour réduites au minimum. Puis, après que la partie « Jeu proprement dit » du

```
# Le joueur désire-t-il faire une autre révision de la zone de travail ?

çava  GOSUB effacécran
      PRINT " ENCORE ? "
      GOSUB uncar                     # réponse mono-caractère
   IF case
          X$ = " Q " THEN OK = – 1    # Q = abandon : ne pas sauvegarder la
                                      zone de travail
          X$ = " Y " THEN OK = 0      # Y = oui, faire une autre passe
      else
          OK = 1                      # autrement c'est OK ; terminé pour les
                                      modifs
        RETURN

510  GOSUB2200:PRINT"ENCORE?";:GOSUB2180:OK=1
520  IFX$="Q"THENOK=–1
530  IFX$="Y"THENOK=0
540  RETURN
```

La routine « çava » demande au joueur si des mises au point supplémentaires sont nécessaires. Si le joueur répond « Y », la routine retourne OK avec la valeur 0, ce qui signifie au programme appelant qu'un appel supplémentaire à « modifs » est demandé. Si le joueur répond par la lettre " Q", la routine retourne OK = – 1 ce qui veut dire que le processus de mise à jour est interrompu et que le contenu de la zone de travail doit être abandonné sans être recopié dans le fichier « joueurs ». Toute autre réponse entraîne un retour avec OK = 1. Dans ce dernier cas le programme appelant passera la main à « misajour » ou à « stockernouveau » pour sauvegarder le contenu de la zone de travail dans la zone de données joueurs.

Ces choix très simples proposés par cette routine sont essentiels au déroulement harmonieux du cycle « d'édition ».

Figure 7.25 : Avez-Vous « Encore » des Modifications ?

programme ait été détaillée, les souches chargées des mises à jour sont devenues les versions montrées aux figures 7.45 à 7.56. Etant donné qu'une possibilité d'entrer les données « questions » et « joueurs » était dès le début, disponible pour permettre les vérifications, les souches simulant le chargement de données enregistrées sur mémoires externes, n'ont pas eu à être développées pour créer des fichiers de données factices d'essai.

Techniques générales de programmation

Nous venons de voir des principes généraux qui s'appliquent à tous les cas de programmation. Il existe également, un certain nombre de techniques de réalisation des programmes qui sont illustrées dans Jumelage et s'appliquent à un large éventail de situations.

Conception de la documentation de l'utilisateur et du programmeur. Vous serez peut-être surpris d'apprendre que la documentation de l'utilisateur et celle du programmeur, sont une

Etapes de la Création et de la Documentation du Jeu du Jumelage

- L'auteur a conçu et affiné le principe original. La définition définitive du comportement du programme fut arrêtée avant le début de la moindre réalisation du programme.
- Les descriptions en BASIC Libre des figures 7.16 à 7.59 ont, dans un premier temps été écrites avec des « souches » beaucoup plus simples pour les routines de création de questions et d'édition-mise à jour des figures 7.45 à 7.56.
- La traduction (« manuelle ») du BASIC Libre en BASIC a été réalisée.
- Le programme a été mis au point et des améliorations ont été apportées aux sorties sur écran et au dialogue. Les souches ont été remplacées par des routines réelles de mise à jour. Toutes ces modifications ont été faites au niveau du BASIC Libre puis incorporées au programme BASIC.
- Un listing BASIC fut tiré, et BASIC et BASIC Libre furent combinés pour former les figures 7.16 à 7.59. C'est alors que les commentaires des figures furent écrits ; et cet examen attentif mit en lumière la nécessité de quelques modifications mineures. Celles-ci furent apportées d'abord au BASIC Libre, puis au BASIC.
- Le programme fut exécuté avec un jeu d'essai. Au cours de cet essai les figures 7.1 à 7.15 ainsi que le texte d'accompagnement furent préparés. Ceci amena d'autres changements dans le programme et les figures 7.16 à 7.59 furent modifiées en conséquence.

Figure 7.26 : Création et Documentation de Jumelage

des premières tâches que vous devez effectuer quand vous réalisez un programme. Malheureusement, cette tâche est souvent retardée jusqu'au dernier moment et exécutée sans enthousiasme.

La figure 7.26 montre les étapes suivies lors de la réalisation du programme et de la documentation du jeu du Jumelage. Vous remarquerez que le programme et la documentation ont avancé en parallèle sans qu'on les laisse diverger.

La raison principale du développement du BASIC Libre a été la nécessité de libérer le programmeur de l'utilisation des numéros de lignes. Le processus montré en figure 7.26, montre combien cet objectif était essentiel et le succès avec lequel il a été atteint. Ce processus de développement implique qu'il y aura des modifications

```
# Routine de modification de nom
changernom  GOSUB effacecran
            IF WN$ = "" THEN {                  # si pas de nom dans fichier
                PRINT "NOM :";                  # en saisir un
                repeat {
                    GOSUB entréechaîne
                    WN$ = SS$
                    } until (WN$ <> "")
                }
          else                                  # si nom dans fichier
                PRINT WN                        # l'afficher
                PRINT "NOM :";
                GOSUB entreechaine              # saisir nouveau nom
                IF SS$ <> "" THEN
                    WN$ = SSS$                  # " RETURN " ne rien changer
                }
            RETURN

550  GOSUB2200:IFWN$<>"" THEN590
560  PRINT"NOM:";
570  GOSUB610:WN$=SS$:IFWN$=""THEN570
580  RETURN
590  PRINTWN$:PRINT"NOM:";:GOSUB610:IFSS <>""THENWN$=SS$
600  RETURN
```

La routine « changernom » affiche le nom en cours et saisit le nom de remplacement éventuel (ou " RETURN " s'il n'y a rien à changer).

Figure 7.27 : Création ou Modification du Nom d'un Joueur

```
# Routine d'entrée d'une chaîne — SS$ = tous caractères tapés jusqu'au premier
RETURN

entreechaine  SS$ = ""                                    # initialiser SS$
              repeat {
                  GOSUB uncar                             # attendre un carac-
                                                          tère
                  IF X$ = " return " THEN break           # terminé si le carac-
                                                          tère est RETURN
                  else IF X$ <> "effacement" THEN {       # si autre qu'efface-
                                                          ment
                      PRINT X$;                           # répercuter/écran
                      SS$ = SS$ + X$                      # concatener à SS$
                      }
                  else IF LEN(SS$)<> 0 THEN {             # s'il en reste au
                                                          moins un
                      PRINT " chaine d'effacement ";      # l'effacer de l'écran
                      SS$ = LEFT$(SS$, LEN(S$S)-1)        # l'enlever de SS$
                      }
                  }
              IF SS$ = " Q " THEN                         # effacer la lettre "Q"
                  PRINT " chaîne d'effacement "
              else IF SS$ <> "" THEN PRINT                # répercuter RETURN
                                                          si chaîne non vide
              RETURN
```

```
610    SS$=""
620    GOSUB2180:IFX$=CHR$(13)THEN660
630*   IFX$<>CHR$(20)THENPRINTX$;:SS$=SS$+X$:GOTO620
640**  IFLEN(SS$)<>0THENPRINTX$;:SS$=LEFT$(SS$,LEN(SS$)-1):GOTO620
660*   IFSS$="Q"THENPRINTCHR$(20);:RETURN
665    IFSS$<>""THENPRINT
670    RETURN
```

La routine « entréechaîne » est semblable dans ses fonctions à l'instruction LINE INPUT SS$. Quand il traite le caractère d'effacement (RUBOUT, flèche arrière, DEL). Le programme doit vérifier qu'il reste des caractères à effacer dans SS$. La programmation correspondante est plus compliquée dans la version Apple car l'expression LEFT$(SS$, 0) est illégale en BASIC Apple. (Voir figure 4.10.)

La routine a deux particularités qui la distinguent de LINE INPUT. Si SS$ = « Q », la lettre Q est alors effacée de l'écran. Si SS$ = « », le caractère RETURN n'est pas répercuté. Dans le cas contraire, le retour à la ligne est assuré par une instruction PRINT.

---

* Dans la version Apple, CHR$(8) est le caractère d'effacement, et CHR$(8) ; «» ; CHR$(8), la chaîne d'effacement. Sur TR$-80, CHRS(8) représente la chaîne d'effacement.

** Voir notes de la figure 4.10 pour la version Apple de cette ligne.

Figure 7.28 : Entrée d'une Chaîne de Caractères

```
# Mise à jour des réponses pour un membre du groupe GP ; GC est l'autre groupe

changechoix  FOR QX = 1 TO NQ(GP)
                 GOSUB effacecran
                 GQ = GP : GOSUB montrerquestion        # afficher la question
                 PRINT " CHOIX:";
                 IF WA(QX) < > 0 THEN                   # afficher le choix (la
                                                        réponse choisie) s'il y
                                                        en a un
                     PRINT WA(QX) ; « : »
                 repeat {                               # saisir Q, D, RETURN
                                                        ou
                     GOSUB uncar                        # un numéro de choix
                                                        valide
                     IF case
                         X$ = " Q " or " return " THEN  # Q = mise à jour ter-
                                                        minée
                             break                      # RETURN = ques-
                                                        tion suivante
                         X$ = « D » THEN }              # D = annuler choix
                             WA(QX) = 0
                             break
                             }
                     else {                             # si choix valide
                                                        # prendre ce choix
                         NN = VAL(XS)
                         IF 1 < = NN < = NC(QX,GP) THEN
                             WA(QX) = NN
                             break
                             }
                         }
                     }
                 IF X$ = " Q " THEN
                     break
                 NEXT QX
             RETURN
```

La routine « changechoix » affiche la réponse courante (s'il y en a une) donnée par le joueur à chacune des questions. Elle attend alors une entrée. La touche RETURN signifie qu'on ne modifie pas ; un D annule la réponse antérieure, un Q veut dire qu'on en a terminé et un numéro de choix (réponse) valide entraîne l'enregistrement de ce choix.

Figure 7.29 : Modification des Réponses aux Questions

à apporter au programme après la rédaction des premières descriptions. Ces modifications peuvent être effectuées sans difficultés, car le programme BASIC peut être renuméroté et les instructions BASIC qui figurent dans les descriptions, peuvent être remplacées sans interférer sur quoi que ce soit dans le reste des descriptions ou dans le commentaire d'accompagnement. La conception du programme, sa réalisation et la documentation ont été totalement libérées de l'utilisation de numéros de lignes. Les numéros de lignes ont servi uniquement pour transcrire le BASIC Libre en BASIC, processus mécanique qui, dans une étape ultime, sera peut-être effectué par ordinateur. Le but des instructions en BASIC réel que l'on trouve aux figures 7.16 à 7.59, est de donner des exemples concrets de la traduction de BASIC Libre en BASIC. Autrement, ces instructions BASIC, comme le code objet qui accompagne un listing FORTRAN ou PL/I, ne sont nécessaires que pour rendre plus aisés, certains cas de mise au point d'ailleurs peu fréquents.

L'étude ci-dessus a mis l'accent sur l'importance du langage de programmation dans l'élaboration de la documentation. De la même façon qu'une documentation « utilisateur » est d'autant plus efficace qu'elle s'appuie sur des exemples réels du comportement du programme, de même la documentation « programmeur » est d'autant plus utile qu'elle s'appuiera sur les listings du programme. BASIC n'ayant aucune façon de mettre en évidence la structure du programme et ne fournissant pas un code source très compréhensible, la technique de description des programmes en BASIC Libre a reçu dans ce livre le rôle de langage source. En conséquence, toutes les modifications ont d'abord été apportées à la description en BASIC Libre, avant d'être exécutées sur le programme réel BASIC. Cela

```
680   FORQX=1TONQ(GP):GOSUB2200
690*  GQ=GP:GOSUB1240:PRINT"CHOIX:";:IFWA(QX)<>0THENPRINTWA(SQX);":";
700   GOSUB2180
710   IFX$="Q"ORX$=CHR$(13)THEN750
720   IFX$="D"THENWA(QX)=0:GOTO750
730   NN=VAL(X$):IF1<=NNANDNN<=NC(QX,GP)THENWA(QX)=NN:GOTO750
740   GOTO700
750   IFX$="Q"THENRETURN
760   NEXTQX:RETURN
```

* La version Apple de cette ligne n'en diffère que par le nombre d'espaces dans les constantes littérales.

Figure 7.29a : Modification des Réponses aux Questions *(BASIC Réel)*

signifie que la documentation du programmeur dont le centre est la description en BASIC Libre, reflète en permanence l'état du programme. C'est un des avantages qu'un langage de programmation auto-documenté a sur des outils de description comme les organigrammes : toute modification doit d'abord être réalisée au niveau de la documentation, avant de pouvoir être apportée au programme.

Conception des interactions homme machine. Le point suivant figurant sous le titre « Techniques Générales de Programmation » porte sur la conception des interactions de la machine avec l'utilisateur. Depuis le milieu des années 60, quand l'utilisation de systèmes interactifs commença à se développer, l'étude de l'interaction qui a lieu entre l'utilisateur (opérateur) et le programme, a été largement négligée. De nombreux programmes, par ailleurs correctement conçus, ont été parsemés d'entrées-sorties conversationnelles, disposées au petit bonheur par le programmeur comme un simple sous-produit des parties des programmes qui effectuent le traitement. D'autres programmes ont des interactions qui ont pris pour modèle, le genre d'entrées-sorties approprié aux systèmes non interactifs. D'autres encore, ont été conçus pour des terminaux imprimants comme les télétypes ; les propriétés des consoles d'affichage sur écran sur lesquelles ces programmes sont à l'heure actuelle utilisés, restent ignorées.

Une faute communément rencontrée dans la programmation conversationnelle, réalisée par des programmes d'ailleurs consciencieux et désireux de bien faire, est le surcroît d'assistance fourni. Les messages explicites, les offres de guidage, les menus et autres distractions du même genre, sont utiles pour le joueur novice, mais tout cela fait écran entre le joueur et la possibilité de faire avancer le jeu de façon vive et sans en être distrait. Leur effet sur un jeu ressemble à celui qu'aurait dans un théâtre, la décision de laisser les lustres allumés pendant toute la représentation, de façon que le public puisse consulter son programme à tout moment.

En étudiant le programme du Jumelage, vous noterez la fréquente utilisation de signaux brefs, de l'effacement de l'écran et de la technique d'entrée « mono-caractère ». Retenez également la routine *entréechaîne* (fig. 7.28) et la façon dont elle est utilisée dans les parties interactives. Le but principal de cette routine est de pallier l'absence de l'instruction LINE INPUT sur la plupart des systèmes BASIC individuels. Elle a cependant deux caractéristiques dont l'origine est l'expérience :

— Effacement du caractère isolé « Q ».

— Pas d'affichage en retour sur une entrée de chaîne nulle (c'est-à-dire qu'un RETURN non précédé d'autre caractère restera sans effet sur l'écran).

Dans le mode conversationnel sur écran, les informations ne doivent pas disparaître avant que le joueur n'ait eu une chance de les lire. Ceci arrive quand les informations à afficher, occupent plus de place que celle qui est disponible sur l'écran (voir la routine *identités* à la figure 7.44) ou bien encore parce qu'on opère un efface-

```
# Demander les préférences du joueur (décondensées dans le tableau FT)
demanderpref  FOR FX = 1 TO NF
                 PRINT : PRINT " VALEUR " ; FV(FX) ;" CHOIX ";
                 IF FT(FX) <> 0 THEN
                    PRINT FT(FX) ;
                 PRINT " : ";
                 repeat {
                    GOSUB uncar
                    IF X$ = " Q ", " N " or " return " THEN
                       break
                    else IF X$ = " D" THEN {     # D = annuler préférence
                       PRINT X$;                 # répercuter D
                       FT(FX) = 0
                       break
                       }
                    else {                       # un nombre exprime une
                                                 préférence
                       NN = VAL(X$)
                       IF 1 < = NN < = MC THEN {
                          PRINT X$;              # répercuter ce choix
                          FT(FX) = NN
                          break
                          }
                       }
                    }
                 IF X$ = " Q" or " N" THEN       # Q = terminé pour la mise à
                                                 jour
                    break                        # N = question suivante
                 NEXT FX
              RETURN
```

La routine « demanderpref » affiche, à partir du tableau FT, les différentes préférences en cours, et saisit les modifications ou les additions. Q, D et RETURN sont interprétés comme dans « changechoix »; N signifie passage à la question suivante.

Figure 7.30 : Création ou Modification des Préférences du Joueur

ment de l'écran avant l'entrée d'une nouvelle commande. Les routines 7.18 *nouveaujoueur* et 7.22 *stockernouveau* ainsi que la routine *jumelage* à la figure 7.33, donnent des exemples de ce problème.

Une autre caractéristique importante des programmes interactifs (caractéristique utilisée lors des modifications et mises à jour), est la possibilité de se dégager d'une transaction, soit parce qu'on s'y est engagé par erreur, soit parce qu'on a commis des erreurs en l'exécutant, ou encore simplement parce que le joueur a décidé de ne pas aller jusqu'au bout. Par exemple, notons que la routine *numquest* (fig. 7.56) peut renvoyer QQ = 0 pour indiquer au programme appelant, que le joueur a décidé d'arrêter. Pour bien comprendre l'importance de ce point, imaginez une version de *numquest* qui redemanderait indéfiniment le numéro de la question jusqu'à en obtenir un qui soit valide. Cette solution serait extrêmement démoralisante pour le joueur qui a, par inadvertance, appuyé sur la touche « D » (suppression (delete) d'une question) et se trouve confronté au signal d'appel " NUMERO DE QUESTION : ". Etant donné la conception de *numquest*, le joueur n'a qu'à appuyer sur RETURN pour mettre fin au processus de suppression.

Le programme de Jumelage utilise plusieurs signaux d'appel (prompts). C'est ainsi, par exemple, que lorsqu'il répond à une question et qu'il demande une opération de « jumelage », le joueur n'a pas accès aux fonctions d'édition et de mise à jour pour l'autre groupe. Ces précautions sont efficaces contre l'entrée par inadver-

```
770* FORFS=1TONF:PRINT:PRINT"VALEUR";FV(FX);" CHOIX";
780  IFFT(FX)<>0THENPRINTFT(FX);
790  PRINT": ";
800  GOSUB2180:IFX$=CHR$(13)ORX$="Q"ORX$="N"THEN850
810  IFX$="D"THENPRINTX$;:FT(FX)=0:GOTO850
820  NN=VAL(X$)
830  IF1<=NNANDNN<=MCTHENPRINTX$;:FT(FX)=NN:GOTO850
840  GOTO800
850  IFX$="Q"ORX$="N"THEN870
860  NEXTFX
870  RETURN
```

* La version Apple de cette ligne n'en diffère que par le nombre d'espaces dans les constantes littérales.

Figure 7.30a : Création ou Modification des Préférences du Joueur *(BASIC Réel)*

tance, de commandes non prévues. Cependant, il est impossible de se prémunir contre les manœuvres intentionnelles. Tout joueur placé sous contrôle d'un signal donné, peut en changer à n'importe quel moment puisque les commandes qui permettent ce transfert, ne sont pas secrètes et ne sont pas protégées de façon spéciale. Dans un seul cas (voir figure 7.16) le programme demande " VRAIMENT ? " et attend qu'on lui réponde oui " Y "), avant d'activer une commande qui peut s'avérer dangereuse.

Traitement des erreurs. Le traitement des erreurs est effectué dans le programme du Jumelage de façon subtile, en évitant les messages d'erreur voyants qui cassent le cours du jeu. La plupart des entrées mono-caractères erronées est simplement ignorée et n'est pas renvoyée sur l'écran. La figure 7.27 *changernom* montre

```
# Misajour des préférences d'un joueur de groupe GP ; l'autre est GC
changepref  FOR QX = 1 TO NQ(GC)
                GOSUB effacecran
                CQ = GC : GOSUB montrerquestion       # afficher question et choix
                WC = WW(QX) : GOSUB expansepref       # préférences en cours vers FT
                GOSUB demanderpref                    # add/modif
                GOSUB condensepref : WW(QX)=WC        # remettre dans WC
                IF X$ = " Q " THEN                    # Q = mise à jour terminée
                     break
                NEXT QX
            RETURN

880  FORQX=1TONQ(GC):GOSUB2200
890  GQ=GC:GOSUB1240:WC=WW(QX):GOSUB920:GOSUB770:GOSUB950:
     WW(QX)=WC
900  IFX$<>"Q"THENNEXTQX
910  RETURN
```

La routine « changepref » déroule les questions du groupe GC affichant pour le joueur concerné, les préférences en cours (s'il y en a) à partir du tableau WW de la zone de travail et saisit les modifications et additions.

Les routines « expansepref » et « condensepref » sont appelées pour passer du format condensé utilisé dans le tableau WW au format étendu utilisé par « demandepref » pour traiter les préférences dans le tableau FT.

Figure 7.31 : Modification des Préférences du Joueur

comment l'entrée illégale d'une chaîne de longueur nulle est traitée (et montre pourquoi *entréechaîne* ne répercute pas à l'écran le RETURN après une chaîne vide). La figure 7.22 *stockernouveau* montre de quelle mauvaise façon, on peut s'y prendre pour traiter une erreur fréquente.

Techniques particulières de programmation

En plus des techniques générales, dont il a été débattu dans la section précédente, plusieurs techniques particulières de program-

```
# Routines de transformation des préférences entre la forme condensée WC et le
tableau FT

expansepref     FOR FX = 1 TO NF
                     FT(FX) = WC mod (MC + 1)
                     WC = (WC − FT(FX))/(MC + 1)
                     NEXT FX
                RETURN
condensepref    WC = 0 : FM = 1
                FOR FX = 1 TO NF
                     WC = WC + FT(FX) * FM
                     FM = FM * (MC + 1)
                     NEXT FX
                RETURN

920  FORFX=1TONF:FT(FX)=WC−(MC+1)*INT(WC/MC+1))
930  WC=(WC−FT(FX))/(MC+1):NEXTFX:RETURN
950  WC=0:FM=1:FORFX=1TONF:WC=WC+FT(FX)*FM:FM=FM*(MC+1):NEXT FX
960  RETURN
```

Les routines « expansepref » et « condensepref » assurent l'aller-retour entre les formes condensées et étendues d'un ensemble de préférences concernant une question. Sous la forme étendue, chaque préférence est un nombre compris entre 0 et MC. Un zéro veut dire qu'il n'y a pas de préférence. Une valeur n (1 < = n < = MC) indique le choix n. Il y a « numjuste » préférences, chacune ayant une valeur laquelle est stockée dans le tableau FV. Quand on calcule la valeur numérique d'un jumelage, chaque réponse reçoit une note égale au poids de la question (pris dans le tableau WT) multiplié par la valeur (tiré du tableau FV) correspondant à la première position à laquelle la réponse apparaît dans le tableau des préférences. Si la réponse ne figure pas dans le tableau des préférences elle est notée zéro.

Dans le format condensé, les préférences, qui sont des nombres compris entre 0 et MC, sont traitées comme des chiffres dans un système de numération à base MC + 1. Ce qui veut dire que chaque préférence est multipliée par une puissance de MC + 1, et que les résultats sont alors additionnés.

Figure 7.32 : Compression et Expansion des Préférences

Trouver les meilleurs jumelages pour un joueur

```
jumelage  GOSUB identité                      # coordonnées joueur (PP, GP)
          GOSUB autregroupe                   # autre groupe = GC
          GOSUB effacecran
          GOSUB effacerscore                  # initialisation « scores maxi »
          PW = PP : GW = GP : GA = GC         # préférences du joueur, réponses
                                                des autres
          FOR PX = 1 TO NP(GC)
               PA = PX : GOSUB score          # jumeler (PW, GW) et (PA, GA)
                    (sans changement Ndt)
               GOSUB sauverscore              # sauver scores maxi
          NEXT PX
          PRINT GN$(GC) ; " QUE "; NM$(PP, GP) ; " PREFERERAIT "
          GS = GC : GOSUB afficherscores      # afficher les scores maxi
          GOSUB effacerscore                  # initialisation « scores maxi »
          PA = PP : GA = GP : GW = GC         # réponses du joueur, préfé-
                                                rence des autres
          FOR PX = 1 TO NP(GC)
               PW = PX; GOSUB score           # jumeler (PW, GW) et (PA, GA)
               GOSUB sauverscore              # sauver scores maxi
          NEXT PX
          PRINT : PRINT GN$(GC) ; " QUI PREFERAIENT "; NMS(PP, GP)
          GS = GC : GOSUB afficherscores      # afficher les scores maxi
          GOSUB uncar : RETURN                # laisser affiché jusqu'à touche
                                                appuyée
```

```
970   GOSUB1310:GOSUB1290:GOSUB2200:GOSUB1110
980   PW=PP:GW=GP:GA=GC
990   FORPX=1TONP(GC):PA=PX:GOSUB1050:GOSUB1140:NEXT PX
1000  PRINT GN$(GC);"QUE"; NM$(PP, GP);"PREFERERAIT":GS=GC:GOSUB1120
1010  GOSUB1110:PA=PP:GA=GP:GW=GC
1020  FORPX=1TONP(GC):PW=PX:GOSUB1050:GOSUB1140:NEXT PX
1030  PRINT:PRINTGN$(GC);"QUI PREFERERAIENT";NM$(PP,GP):GS=GC:GOSUB1120
1040  GOSUB2180:RETURN
```

La routine « jumelage » trouve quels sont les membres de l'autre groupe qui correspondent le mieux aux préférences du joueur concerné et quels sont ceux dont les préférences sont le mieux satisfaites par ce même joueur. La routine « score » évalue chaque association, tandis que « sauverscore » maintient sous forme triée un tableau des meilleurs scores, et que « afficherscores » affiche les noms des joueurs de l'autre groupe dont les noms se trouvent dans le tableau des scores maxi.

Figure 7.33 : La Formation des Couples

```
# Calculer score SC des réponses du joueur (PA, GA) confrontées aux préférences de
(PW, GW)

score   SC = 0                             # initialisation du score
          FOR QX = 1 TO NQ(GA)             # parcourir les questions du groupe GA
          WC = W(QX, PW, GW)               # préférence de (PW, GW)
          AV = A(QX, PA, GA)               # réponse de (PA, GA)
          GOSUB correspondance
          SC = SC + FT * WT(QX, GA)        # mettre à jour le score
          NEXT QX
        RETURN

1050 SC=0:FORQX=1TONQ(GA)
1060 WC=W(QX,PW,GW):AV=A(QX,PA,GA):GOSUB1080
1070 SC=SC+FT*WT(QX,GA):NEXTQX:RETURN
```

La routine « score » appelle, à répétition, le sous-programme « correspondance » pour associer réponses et préférences et cumule le score SC en additionnant les correspondances affectées du poids.

Figure 7.34 : Evaluation Réponses — Préférences

```
# Evaluer la correspondance FT de la réponse AV confrontée aux préférences codées WC

correspondance  FT = 0                       # initialisation de la correspon-
                                               dance
                GOSUB expansepref            # passer de WC au tableau FT
                FOR FX = 1 TO NF             # parcourir les tableaux FT, FV
                    IF FT(FX) = AV THEN {    # si AV a reçu une valeur
                        FT = FV(FX)          # alors FT = cette valeur
                        break
                        }                    # autrement FT reste à valeur ini-
                                               tialisation
                    NEXT FX
                RETURN

1080 FT=0:GOSUB920
1090 FOR FX=1TONF:IFFT(FX)=AVTHENFT=FV(FX):RETURN
1100 NEXTFX:RETURN
```

La routine « correspondance » trouve (si elle existe) la première occurrence de la réponse AV dans les préférences encodées en WC. Les préférences encodées WC sont transformées par « expansepref » et mises dans le tableau FT. C'est alors que AV est comparé successivement aux éléments de ce tableau.

Figure 7.35 : Evaluation d'une Réponse

mation sont également illustrées par Jumelage. Ces techniques peuvent être utilisées dans de nombreux programmes de différente nature.

Conception des fonctions de mise à jour. Deux genres d'« editing » sont mis en œuvre dans Jumelage. Le premier est montré par les commandes qui sont en figure 7.4. C'est là, une approche habituelle pour éditer et mettre à jour des informations consécutives composées d'éléments numérotés (par exemple, une « page » de texte peut être composée de n lignes, numérotées de 1 à n, à des fins de mise à jour). Les fonctions essentielles sont l'addition, la suppression ou la modification d'un élément. L'addition se fait par l'une ou l'autre de deux fonctions disjointes : insertion d'un élément ou bien addition d'un élément à la fin. Cette séparation en deux fonctions est due au fait qu'il faut trouver une façon de préciser la place de l'élément à insérer. Si la commande d'insertion demande qu'on fixe le rang occupé par l'élément avant lequel il convient d'insérer, il faut alors trouver une formulation particulière pour dire « insertion à la fin » (Voyez-vous pourquoi ?). D'un autre côté, si la commande d'insertion demande l'élément après lequel l'élément nouveau est à placer, un problème se posera dans le cas où l'on désire insérer avant le premier élément. La solution la plus facile est celle qui est utilisée dans le jeu du Jumelage.

La deuxième procédure de mise à jour, utilisée dans Jumelage, est explicitée par le dialogue présenté à la figure 7.15. Lorsque vous désirez mettre à jour vos choix (les réponses faites aux questions) et vos préférences, le programme les affichera un par un, en vous montrant ce qui se trouve dans le fichier pour chaque élément et en vous autorisant à le modifier si nécessaire. Si « pas de réponse » est licite, on peut alors considérer que la modification d'une réponse donnée peut aussi consister à la supprimer sans la remplacer. Cette fonction de suppression est l'objet de la commande « D » de la

Initialiser le tableau des scores maxi

effacerscores SX = 0 : RETURN # " mettre indice tableau à zéro "

```
1110 SX=0:RETURN
```

La routine « effacerscores » initialise les tableaux associés TS, TI en mettant à zéro l'index SX du « dernier » élément.

Figure 7.36 : Mise à Zéro des Meilleurs Scores

figure 7.15. Ce type d'annulation est complètement différent de celui qui correspond à la commande « D » utilisée dans la mise à jour des questions. Quand on met à jour les questions, on est libre de modifier la structure du fichier des questions en même temps que son contenu. Pour les choix (réponse choisie) et les préférences, vous ne pouvez pas toucher à la structure du fichier ; vous pouvez seulement en modifier le contenu. Ce dernier mode de mise à jour est souvent pratiqué par les logiciels interactifs de mise au point — programmes qui vous permettent de voir et de modifier le contenu de la mémoire de l'ordinateur — car il n'est évidemment pas possible de changer les adresses des emplacements mémoire ou le nombre de ces emplacements.

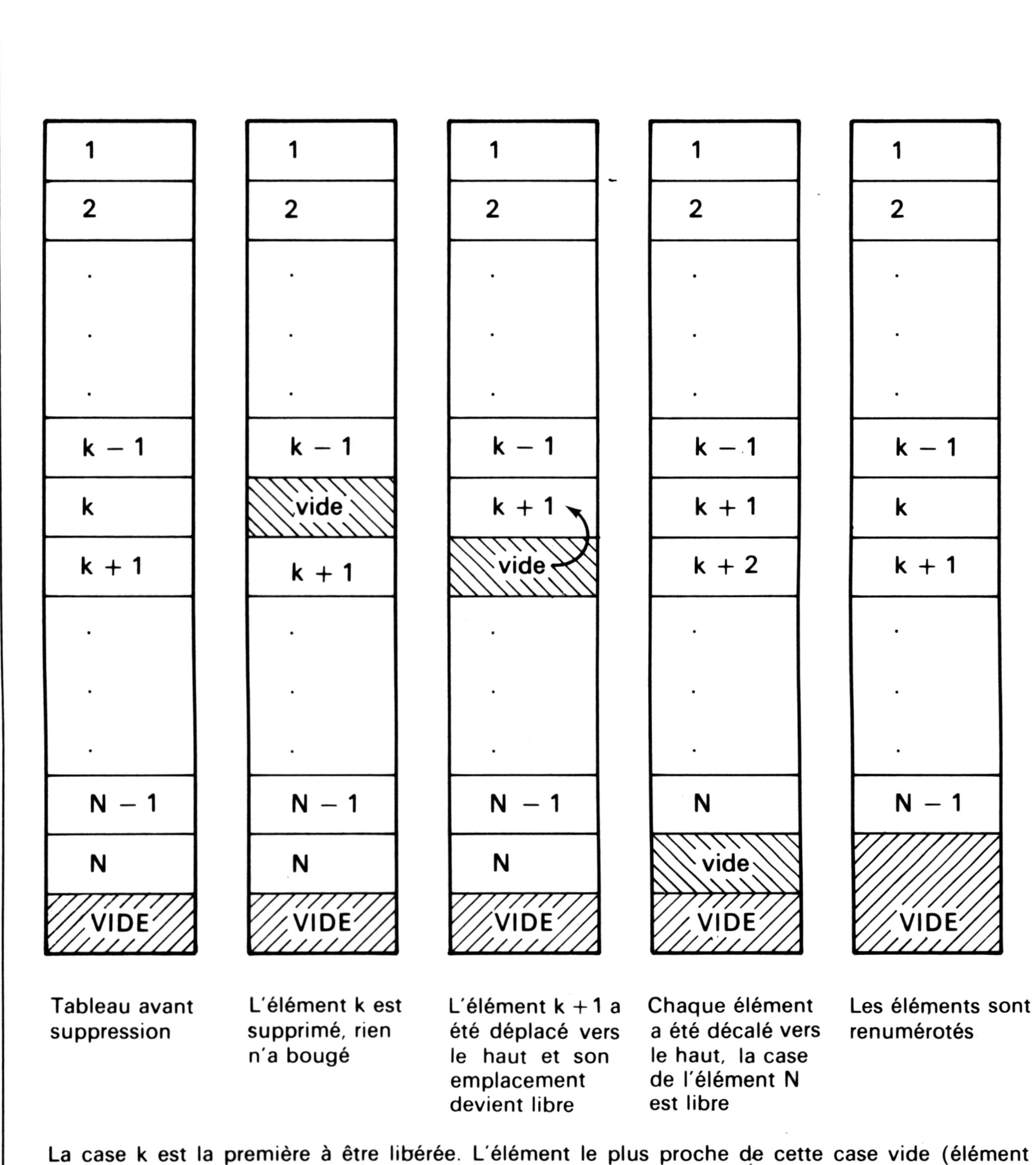
Ce type de mise à jour ne peut être utilisé que pour les fichiers comportant un petit nombre d'éléments, puisque vous devez examiner les éléments un par un jusqu'à ce que vous ayez localisé celui que vous désirez modifier. Il faut donc que le joueur puisse passer rapidement d'un élément à l'autre, faute de quoi le procédé est lassant. Cette progression rapide est rendue possible dans Jumelage par le procédé d'entrée mono-caractère. Il suffit d'appuyer sur RETURN pour laisser un choix ou une préférence en l'état et passer au suivant. Il suffit de taper le numéro de la réponse choisie (il ne peut y en avoir plus de neuf par question) pour modifier la réponse choisie ou la préférence. Vous frappez le caractère « Q » quand vous avez terminé.

```
# Afficher les scores maxi (groupe GS) à partir des tableaux TI, TS
afficherscores  IF SX < = 0 THEN PRINT "AUCUN "          # si tableau vide,
                                                         # ne rien afficher
                else
                    FOR SZ = 1 TO SX                     # parcourir les tableaux
                        PRINT TS(SZ), NM$(TI(SZ), GS)    # afficher le score et le
                                                         # nom du joueur
                        NEXT SZ
                RETURN

1120  IFSX<=0THENPRINT"AUCUN":RETURN
1130  FORSZ=1TOSX:PRINTTS(SZ),NM$(TI(SZ),GS):NEXTSZ:RETURN
```

La routine « afficherscores » parcourt les deux tableaux TI et TS affichant le score à partir de TS et le nom du joueur en se servant de l'indice pris dans TI et du numéro de groupe GS.

Figure 7.37 : Affichage des Joueurs ayant les Meilleurs Scores



La case k est la première à être libérée. L'élément le plus proche de cette case vide (élément k + 1) est décalé en premier.

Figure 7.38 : Suppression de l'Elément k

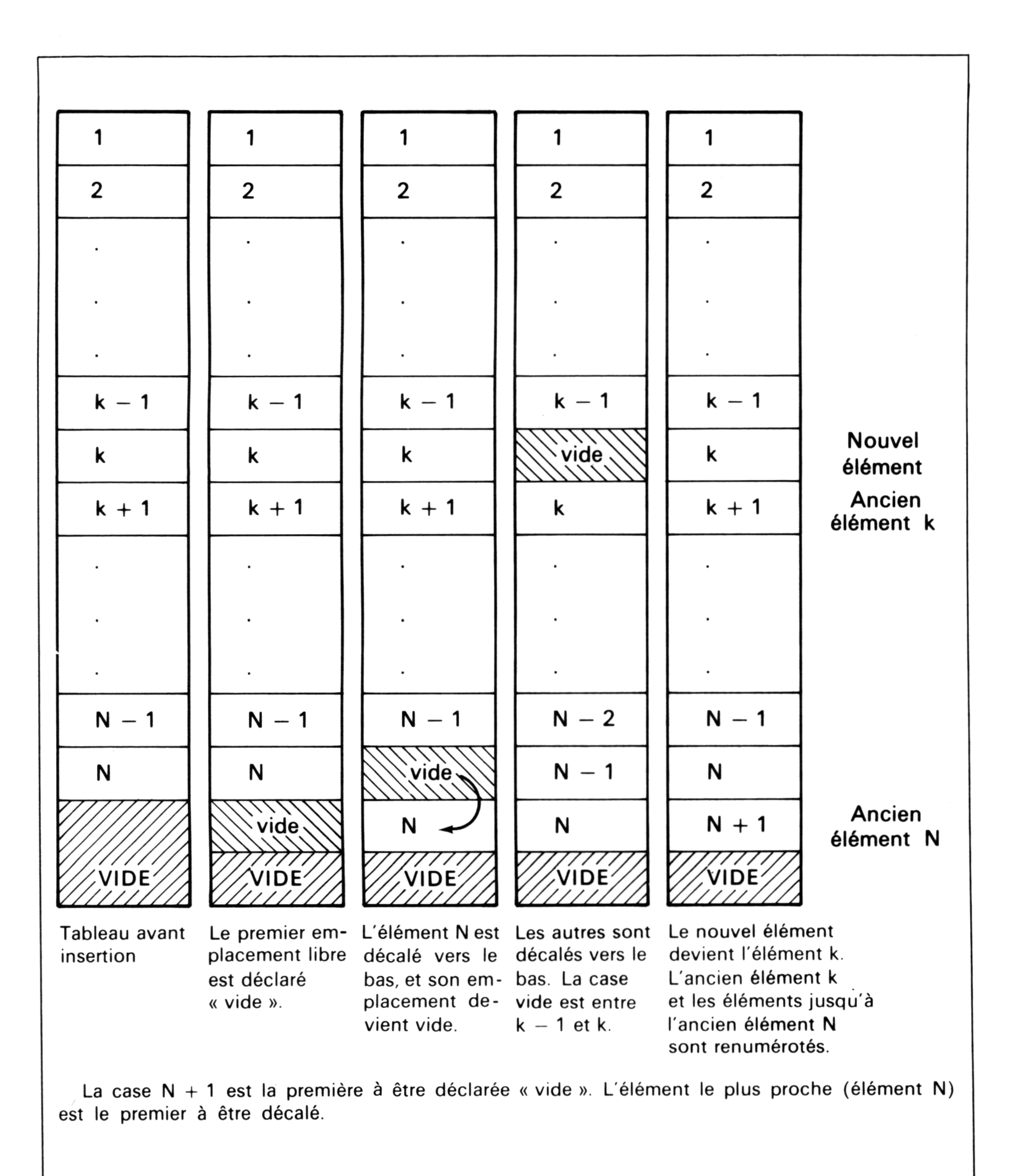


Figure 7.39 : Insertion Devant l'Elément k

Algorithmes d'insertion et de suppression. Les figures 7.52 et 7.55 donnent des exemples de suppressions. Les figures 7.40 et 7.53 montrent des insertions. Des algorithmes d'insertion et de suppression similaires trouvent leur emploi chaque fois qu'on a un tableau où l'ordre des éléments a son importance. Quand on ajoute un élément, ce dernier doit être mis dans le tableau à la place correspondant à l'indice convenable, et tous les éléments qui viennent ensuite, doivent être décalés. Quand un élément est supprimé, tous les éléments suivants doivent être « déplacés » de façon à combler la case vide.

Regardons ce qui se passe pour les différents éléments du tableau lorsqu'il y a insertion ou suppression. En cas d'insertion, chaque élément (s'il y en a) qui suit l'élément inséré, doit se retrouver avec l'indice immédiatement supérieur; s'il y a suppression, chaque élément (s'il y en a) qui suit l'élément supprimé, doit passer au rang immédiatement inférieur.

Quand un élément d'un tableau est annulé, une case vide est donc créée quelque part et le premier élément qui doit être décalé, est celui dont l'emplacement final est cette place libérée (voir figure 7.38).

Ce « mouvement » crée une autre case vide et l'élément suivant à décaler, est celui qui est contigu à cette case. De cette façon le « trou » se déplace vers la fin de la table et se trouve finalement incorporé à l'espace libre qui, éventuellement, y existe déjà.

Quand un élément doit être inséré (voir figure 7.39), il n'existe aucune case libre correspondant à l'endroit où il doit finalement se trouver (sauf dans le cas où il doit être placé en fin de table). Le premier emplacement vide en fin de table est repéré, et l'élément voisin (c'est-à-dire le dernier élément de la table) y est alors décalé. La case vide se trouve maintenant à l'endroit où se trouvait le dernier élément, et l'avant-dernier élément est décalé d'un cran vers la fin de la table pour combler ce trou. De telle sorte que l'emplacement libre finit par occuper la place où le nouvel élément sera inséré. La renumérotation mentionnée dans les légendes des figures 3.38 et 3.39 se fait automatiquement car les indices de la table concernent les emplacements, non les éléments eux-mêmes.

En résumé, la règle qu'il faut retenir en matière d'insertion et de suppression, est la suivante : l'élément le plus voisin du trou doit être décalé le premier.

Gestion de fichiers. La technique de programmation dont nous allons parler maintenant, est celle de gestion des fichiers. C'est un sujet très vaste et le programme du Jumelage ne l'éclaire que très partiellement. En général, un fichier est un ensemble d'éléments baptisés enregistrements où chacun de ceux-ci contient des infor-

Insérer le score SC et l'indice joueur PX dans les deux tableaux (TS, TI)

```
sauverscore  IF SX = 0 THEN                          # si tableau vide,
                 GOSUB alasuite : RETURN             # insérer « à la fin »
             FOR SZ = 1 TO SX
                 IF SC > TS(SZ) THEN {               # insérer avant SZ
                     IF SX < MT THEN
                         SX = SX + 1                 # il y a la place pour
                                                     un de plus
                     SE = SX - 1                     # plus grand indice
                                                     à décaler
                     IF SE > = SZ THEN
                         FOR SY = SE TO SZ STEP - 1  # décaler plus grand
                                                     indice d'abord
                             TS(SY + 1) = TS(SY)     # chacun est décalé
                                                     d'un cran
                             TI(SY + 1) = TI(SY)
                             NEXT SY
                     TS(SZ) = SC                     # insérer nouvel élé-
                                                     ment
                     TI(SZ) = PX
                     RETURN
                     }
                 NEXT SZ                             # ne pas insérer
                                                     devant SZ
             IF SX < MT THEN                         # s'il y a de la place,
                 GOSUB alasuite                      # insérer à la fin
             RETURN
```

```
1140  IFSX=0THENGOSUB1230:RETURN
1150  FORSZ=1TOSX:IFSC<=TS(SZ)THEN1210
1160  IFSX<MTTHENSX=SX+1
1170  SE=SX-1:IFSE<SZTHEN1200
1180  FORSY= SETOSZSTEP-1
1190  TS(SY+1)=TS(SY):TI(SY+1)=TI(SY):NEXTSY
1200  TS(SZ)=SC:TI(SZ)=PX:RETURN
1210  NEXTSZ:IFSX<MTTHENGOSUB1230
1220  RETURN
```

La routine « sauverscore » parcourt le tableau TS, comparant le score SC aux éléments du tableau TS. Etant donné que TS est en ordre décroissant, la première position où SC devient supérieur à l'élément de TS est l'indice auquel SC et PX devront être insérés dans TS et TI.

Figure 7.40 : Maintien des Scores en Ordre Décroissant

mations spécifiques se rapportant au sujet d'ensemble couvert par le fichier. Par exemple, dans un fichier de noms de clients, numéros de téléphone et pointures de chaussures, un enregistrement pourrait contenir

PAUL BUNYAN 1-7276570 51

Les fichiers de ce genre sont généralement conservés sur une mémoire externe telle que disque ou bande.

Dans Jumelage, les « fichiers » sont les tables contenant les questions et les données concernant le joueur. Par exemple, « l'enregistrement » du joueur 1 du groupe 1 contient :

NM$(1,1)
A(QX,1,1) QX = 1,...,NQ(1)
W(QX,1,1) QX = 1,...,NQ(2)

La chaîne NM$(1,1) est le nom du joueur, le nombre A(QX,1,1) est la réponse du joueur à la question numéro QX du groupe 1 et W(QX,1,1) est un nombre qui représente sous sa forme codée, les préférences du joueur pour les réponses possibles à la question QX du groupe 2. Les expressions QX = 1,...,NQ(1) et QX = 1,...,NQ(2) indiquent qu'il y a une réponse pour chaque question du groupe 1 et une préférence codée pour chaque question du groupe 2. NQ(1) est le nombre de questions du groupe 1, NQ(2), celui des questions du groupe 2.

En ce qui concerne la gestion des fichiers, l'un des principaux objectifs lors de la conception, est la conservation de l'intégrité des données. Ce qui veut dire que les sous-programmes de gestion de fichiers doivent s'assurer que l'information valide est enregistrée à l'endroit valable et que les données fausses ne s'infiltrent nulle part. Il est évident que les routines de gestion de fichiers ne peuvent savoir si le nommé PAUL BUNYAN chausse réellement du 51, mais elles

```
alasuite  SX = SX + 1          # allonger la liste de un élément
          TS(SX) = SC          # ajouter cet élément à la fin
          TI(SX) = PX
          RETURN

   1230   SX=SX+1:TS(SX)=SC:TI(SX)=PX:RETURN
```

La routine « àlasuite » est un petit sous-programme qui n'est appelé que de la routine « sauverscore ».

Figure 7.40a : Un Sous-Programme Appelé par Sauverscore

peuvent au moins faire en sorte que la personne qui entre en mémoire cette information, la trouve à son goût avant qu'elle ne soit enregistrée dans les fichiers. C'est le rôle de la zone de travail utilisée pour la mise à jour des informations concernant le joueur dans le programme du Jumelage.

La routine *editjoueur* de la figure 7.17 montre comment la zone de travail est utilisée : la routine *changerzone* va chercher « l'enregistrement » du joueur considéré (c'est-à-dire le contenu des éléments adéquats des tableaux NM$, A et W) et met ces données dans la zone de travail ; la routine *modifs* permet alors au joueur de modifier les données se trouvant dans la zone de travail (le contenu des tableaux initiaux n'est pas changé pendant cette phase de la mise à jour) ; et finalement, la routine *misajour* est appelée si, et seulement si, le joueur a répondu par " N " à la question " ENCORE ? ". Une réponse " Q " annule la mise à jour en cours, tandis que toute autre réponse permet au joueur de reprendre le cycle des opérations de mise à jour.

Assurez-vous d'avoir bien compris le processus de mise à jour que nous venons de commenter. Tout bon sous-programme d'édition doit vous permettre de « laisser tomber » une mise à jour en cours d'exécution, sans que cela « abime » l'information antérieure. Effectuer les modifications sur un « brouillon de travail » est la principale méthode pour y parvenir. (Essayez d'imaginer une autre procédure.)

En avançant dans l'analyse du programme du Jumelage, il serait utile que vous observiez la façon dont la mise à jour du fichier des questions, a été traitée. Demandez-vous en quoi et pourquoi la

```
#  Afficher la question (QX, GQ) et ses choix

montrerquestion   PRINT Q$(QX, GQ)
                  FOR CX = 1 TO NC(QX, GQ)
                       PRINT CX;" "; C$(CX, QX, GQ)
                       NEXT CX
                  RETURN

   1240   PRINTQ$(QX,GQ)
   1250   FORCX=1TONC(QX,GQ):PRINTCX;" ";C$(CX,QX,GQ):NEXTCX:RETURN
```

La routine « montrerquestion» affiche une question et le choix de réponses correspondant. L'indice de la question est QX, le numéro de groupe GQ.

Figure 7.41 : Affichage d'une Question

technique utilisée diffère de celles que nous avons étudiées jusqu'ici.

Economie de place mémoire. Les techniques de programmation que nous désirons vous exposer en dernier lieu, concernent l'économie de place en mémoire. Il s'agit d'un problème universel ; il n'y a jamais assez de place mémoire. Du plus minuscule microprocesseur au plus énorme des « monstres », tôt ou tard le programme n'a plus assez de place en mémoire. En paraphrasant un principe connu :

> *Le programme croît jusqu'à occuper toute la mémoire disponible.*

En outre (en regardant les choses de façon plus positive), si vous écrivez un logiciel destiné à la commercialisation qu'il soit ou non la totalité du produit proposé, un programme plus petit s'avérera alors moins coûteux et aura un marché potentiel plus ouvert. Par exemple, un programme en BASIC qui tourne avec 8K sur la machine XYZ, sera vraisemblablement acheté en plus grand

```
# Demander quel est le groupe du joueur
demandergroupe  PRINT                                # à la ligne
                FOR GX = 1 TO 2                      # afficher noms des
                                                     groupes
                      PRINT GX;" "; GN$(GX)
                      NEXT GX
                PRINT "GROUPE: "                     # demander le groupe
                                                     du joueur
                repeat {
                      GOSUB uncar
                      GP = VAL(X$)
                      } until (GP = 1 OR GP = 2)
                PRINT X$;                            # répercuter/écran ré-
                                                     ponse joueur
                RETURN

1260  PRINT:FORGX=1TO2:PRINTGX;" ";GN$(GX):NEXTGX:PRINT "GROUPE:";
1270  GOSUB2180:GP=VAL(X$):IFNOT(GP=1ORGP=2)THEN1270
1280  PRINTX$;:RETURN
```

La routine « demandergroupe » affiche les noms des deux groupes, demande le numéro du groupe du joueur et accepte comme réponse soit 1, soit 2.

Figure 7.42 : De quel Genre êtes-vous ?

nombre par des propriétaires de XYZ, que ne le sera un programme donnant les mêmes résultats mais requérant 48K (à moins que tous les modèles XYZ ne possèdent au minimum 48K de mémoire).

Il y a deux façons d'économiser la place mémoire sur les ordinateurs individuels :

— Compression du programme BASIC.

— Programmation économe (de place) et algorithmes de compression de données.

Les occasions que vous aurez de comprimer votre programme BASIC, dépendent des caractéristiques de votre système. Par exemple sur TRS-80, tous les espaces entrés entre le numéro de ligne et la fin de la ligne BASIC, sont inclus dans le programme ; chaque espace (sauf le premier) vous coûte un octet en mémoire. Sur Pet, les espaces compris entre le numéro de ligne et le premier caractère qui ne soit pas un blanc, sont ignorés (au moment du listing, le Pet insère un espace à cet endroit), mais tous les autres espaces présentents dans la ligne feront partie du programme. Sur Apple, tous les espaces tapés sont ignorés ; Apple a ses règles personnelles d'espacement qu'il applique au moment du listing. (Sur tous ces systèmes, chaque espace faisant partie d'une constante littérale — par exemple, " COMMENT ALLEZ-VOUS ? " — occupe un octet, même sur TRS-80 qui a des caractères spéciaux (code ASCII 192 à 255) représentant des blancs consécutifs avec un maximum de 63).

Jumelage a été développé d'abord sur un Pet 8K et les instructions BASIC des figures 7.16 à 7.59 ne comportent pratiquement

```
#  Calculer le numéro GC du groupe « autre » que GP

autregroupe   IF GP = 1 THEN            # si vous êtes 1
                  GC = 2                # l'autre est 2
              else                      # et vice versa
                  GC = 1
              RETURN

   1290  IFGP=1THENGC=2:RETURN
   1300  GC=1:RETURN
```

La routine « autregroupe », à partir d'un groupe GP, calcule le numéro de « l'autre » groupe GC. Notons que cette relation élémentaire ne peut pas être représentée de façon correcte par une fonction BASIC. (On peut « tricher » : DEFNOT(X)=3−X.)

Figure 7.43 : Le Genre des « Autres »

aucun blanc superflu. Bien sûr, leur lisibilité en est quelque peu réduite, mais ce n'est pas important, puisque le BASIC réel n'est pas destiné aux yeux humains. Seule, la description en BASIC Libre est faite pour être lue.

Pour la même raison, l'absence d'instruction REM dans le programme BASIC ne constitue pas un problème car les formes BASIC Libre et BASIC réel sont associées en petits modules dans les figures présentant le programme du Jumelage. En fait, étant donné que c'est vous qui devrez traduire le BASIC Libre en BASIC réel et que vous n'avez pas de programme pour réaliser ce rapprochement à votre place, il pourrait être utile de commencer chaque routine par une instruction REM précisant le nom de celle-ci. De cette façon, si vous avez la possibilité d'afficher toutes les lignes qui contiennent un ou des codes particuliers, vous pourrez générer une « table des symboles » qui fera la liaison numéros des lignes-noms des routines. Par exemple, sur Pet avec la « Panoplie du Programmeur » de Nestar, vous pouvez entrer la commande

```
FIND REM
```

pour obtenir l'affichage suivant

```
210  REM EDITJOUEUR
260  REM NOUVEAUJOUEUR
300  REM EFFACERZONE
          .
          .
          .
```

Si vous vouliez stocker cela sur cassette, vous pourriez le faire par un programme qui le mettrait sous la forme

```
EDITJOUEUR       210
NOUVEAUJOUEUR    260
EFFACERZONE      300
     .
     .
     .
```

Finalement, une fois enregistré sur cassette, mettez ce petit fichier en entrée d'un programme de tri pour obtenir un listing trié par ordre alphabétique qui peut être utile.

```
CAVA                510
DEMANDERGROUPE     1260
DEMANDERPREF        770
     .
     .
     .
```

```
# S'assurer de l'identité du joueur (PP, GP)

identité  GOSUB demandergroupe                     # obtenir le groupe du joueur

          GOSUB effacecran

          IF NP(GP) = 0 THEN {
               PRINT "POUR";GN$(GP) ; " FICHIER VIDE"
               XX = 0 : RETURN                      # retour du signal « échec »
               }

          XX = 1                                    # retour du signal « succès »

          FOR PX 1 TO NP(GP) STEP pagecran
               GOSUB effacecran
               PT = min(PX + pagecran - 1, NP(GP))  # indice maxi pour cette
                                                    page
               FOR PY = PX TO PT                    # afficher une page de
                    PRINT PY ; " ";NM$(PY, GP)      # nombres et de noms
                    NEXT PY
            repeat {
                    PRINT " NUMERO: " ; :           # demander le numéro du
                         GOSUB entreechaîne         joueur
                    PP = VAL(SS$)
                    IF 1 < = PP < = NP(GP) THEN     # numéro valide pour ce
                                                    groupe
                         RETURN
                    else IF PP = 0 THEN             # non numérique
                         break
                    }
               IF X$ = "Q" THEN                     # Q signal d'abandon
                    { XX = 0 : RETURN }             # retour du signal « échec »
               NEXT PX

          PRINT " C'EST TOUT POUR ";               # toutes les pages d'écran
               GN$(GP)                                ont été affichées

          XX = 0 : RETURN                           # retour du signal « échec »
```

La routine « identité » demande le numéro du groupe du joueur et montre les noms et numéros des personnes du groupe à raison d'un écran à la fois. Après l'affichage de chaque page d'écran, le joueur est invité à entrer un numéro. La lettre Q signale l'abandon du processus. Toute autre réponse non numérique entraîne l'affichage de la page suivante.

Figure 7.44 : Vos Papiers S.V.P. *(Identité)*

```
1310  GOSUB1260:GOSUB2200:IFNP(GP)>0THEN1330
1320  PRINT"POUR";GN$(GP);" FICHIER VIDE. ":XX=0:RETURN
1330  XX=1:FORPX=1TONP(GP)STEP10:GOSUB2200
1340  PT=PX+9:IFPT>NP(GP)THENPT=NP(GP)
1350  FORPY=PXTOPT:PRINTPY;" ";NM$(PY,GP):NEXTPY
1360  PRINT"NUMERO:";:GOSUB610:PP=VAL(SSS)
1370  IF1<=PPANDPP<=NP(GP)THENRETURN
1380  IFPP<>0THEN1360
1390  IFX$="Q"THENXX=0:RETURN
1400  NEXTPX
1410  PRINT"C'EST TOUT POUR ";GN$(GP):XX=0:RETURN
```

Figure 7.44a : BASIC Réel pour Identité

```
# Créer les questions et leurs poids
creation  repeat {
              GOSUB effacecran                        # démarrer sur écran propre
              PRINT " : : "; : GOSUB uncar : PRINT XS  # saisir et répercuter commande
              IF case
                  X$ = " I "THEN GOSUB partienouvelle  # démarrage à zéro
                  X$ = " N " THEN GOSUB editnoms       # éditer les noms des groupes
                  X$ = " 1 " THEN GOSUB editun         # éditer les questions du groupe 1
                  X$ = " 2 " THEN GOSUB editdeux       # éditer les questions du groupe 2
                  X$ = " L " THEN GOSUB lecture        # lire une cassette ou disque
                  X$ = " S " THEN GOSUB écriture       # écrire sur cassette ou disque
                  X$ = " Q " THEN break                # que le jeu commence
          }
          RETURN
1420  GOSUB2200:PRINT"::";:GOSUB2180:PRINTX$
1430  IFX$="I"THENGOSUB1510:GOTO1420
1440  IFX$="N"THENGOSUB1700:GOTO1420
1450  IFX$="1"THENGOSUB1740:GOTO1420
1460  IFX$="2"THENGOSUB1750:GOTO1420
1470  IFX$="L"THENGOSUB2210:GOTO1420
1480  IFX$="S"THENGOSUB2220:GOTO1420
1490  IFX$<>"Q"THEN1420
1500  RETURN
```

La routine « création » permet de choisir entre plusieurs commandes permettant la création et la mise à jour des questions.

Figure 7.45 : Menu pour la Phase Préparatoire

Saisir données initiales pour partie

```
partienouvelle  FOR GX = 1 TO 2
                    PRINT : PRINT "GROUPE"; GX; " : "
                    repeat                  # saisir nom du groupe
                        GOSUB entreechaine
                        until (SS$ < > " ")
                    GN$(GX) = SS$
                    NQ(GX) = 0              # initialisation nombre de questions
                    while (NQ(GX) < MQ) {   # demander jusqu'à MQ questions
                        GOSUB effacecran
                        QX = NQ(GX) + 1
                        GOSUB entreequest   # saisir une question et les répon-
                                              ses (choix)
                        IF X$ = " Q " THEN  # plus de questions
                            break
                        else
                            NQ(GX) = QX     # mettre à jour nombre de questions
                        }
                    NEXT GX
                RETURN
```

```
1510   FORGX=1TO2
1520*  PRINT:PRINT"GROUPE";GX;":";
1530   GOSUB610:IFSS$=""THEN1530
1540   GN$(GX)=SS$:NQ(GX)=0
1550   IFNQ(GX)>=MQTHEN1590
1560   GOSUB2200:QX=NQ(GX)+1:GOSUB1600
1570   IFX$="Q"THEN1590
1580   NQ(GX)=QX:GOTO1550
1590   NEXTGX:RETURN
```

La routine « partienouvelle » parcourt la liste des éléments nécessaires à la préparation du jeu et demande au joueur de les fournir.

---

* La version Apple de cette ligne n'en diffère que par le nombre d'espaces.

Figure 7.46 : Préparation d'une Nouvelle Partie

Les techniques utilisables pour compresser les programmes BASIC sont nombreuses. Le manuel BASIC de votre système contient vraisemblablement des indications à ce sujet. Nous allons aborder un autre type d'approche qui est la « programmation à l'économie » d'une part, et les algorithmes de compression de données d'autre part.

Il arrive parfois qu'il n'existe pas de méthode « idéale » pour effectuer tel calcul ou stocker telle information. Mais généralement il y a des « compromis » — par exemple, un type de solution donnera un petit programme allant lentement, tandis que l'autre consistera en un programme encombrant mais performant. Il n'y a pas de formule qui permette de faciliter la prise de décision. En réalité, sauf à réaliser effectivement les deux projets, il est généralement impossible de comparer de façon précise, l'encombrement et la vitesse de traitement de chacune des solutions.

Dans Jumelage, le compromis le plus visible entre les contraintes d'espace et celles de temps, se trouve dans l'encodage des préférences du joueur. Pour chacune des questions faisant partie de la liste du groupe des « autres », un joueur peut affecter une des réponses possibles à chacune des « NF » valeurs de préférence. (NF est une variable dont la valeur est précisée dans un ordre DATA, lui-même situé dans la routine *init* — la valeur de NF est de 4 pour les figures 7.11, 7.13 et 7.15). La méthode la plus facile pour stocker ces préférences, serait d'utiliser un tableau de dimensions NF, MQ, MP, GM. Ce qui revient à dire que le choix (la réponse choisie) serait indicé par le groupe, le numéro de joueur, le numéro de question et le numéro de la valeur de préférence. En admettant les valeurs suivantes : NF = 4, MQ = 6, MP = 12, GM = 2, ce tableau aurait 4 × 6 × 12 × 2 = 576 éléments. Lesquels prendraient 6 838 octets de mémoire sur Apple ou Pet et 5 474 sur TRS-80. La solution utilisée dans le programme du Jumelage est d'encoder toutes les valeurs de préférence concernant une question donnée, en un nombre unique (nous verrons comment bientôt), de sorte que les dimensions du tableau soient simplement MQ, MP, GM. En reprenant les valeurs indiquées plus haut, on obtient un tableau de 144 éléments. Ce tableau prend 1 376 octets sur Apple ou Pet et 1 104 sur TRS-80, soit une économie de place de près de 80 % dans les deux cas.

Les préférences sont compressées en un seul mot (2 octets *n.d.t.*) en les traitant comme des nombres dans le système de numération de base MC + 1, où MC est la variable qui précise le maximum autorisé pour les réponses possibles. Cette valeur, qui est précisée en DATA, ne peut dépasser 9.

```
# Saisir la question QX pour le groupe GX

entreequest  PRINT : PRINT "Q "; QX; " POUR ";        # réclamer la question
             GN$(GX)
             repeat {
                  GOSUB entreechaine                   # saisir la chaîne
                  IF SS$ = "Q" THEN {
                       X$ = "Q"                        # Q = plus de questions
                       RETURN
                       }
                  else IF SS$ <> " " THEN {            # chaîne non nulle néces-
                                                       saire
                       Q$(QX, GX) = SS$
                       break
                       }
                  }
             NC(QX, GX) = 0                            # initialisation nombre choix
             while (NC(QX, GX) < MC) {                 # autoriser jusqu'à MC choix
                  CX = NC(QX, GX) + 1
                  PRINT CX; " ";                       # demander le choix sui-
                                                       vant
                  repeat {
                       GOSUB entreechaine              # saisir chaîne
                       IF SS$ = "Q" THEN
                            { break : break }          # Q = fini pour les choix
                       else IF SS$ <> " " THEN {       # chaîne non nulle néces-
                                                       saire
                            C$(CX, QX, GX) = SS$
                            break
                            }
                       }
                  NC(QX, GX) = CX                      # mettre à jour nombre de
                                                       choix
                  }
             PRINT : PRINT "POID$ :";
             GOSUB entreechaine
             WT(QX, GX) = VAL(SS$)
             RETURN
```

La routine « entréequest » réclame et saisit une question et ses choix. Il n'y a pas de fonction d'édition (mise au point) dans cette routine.

Figure 7.47 :* *Création d'une Question *(et des Réponses qu'on pourra choisir)*

```
1600* PRINT:PRINT"Q";QX;"POUR";GN$(GX)
1610  GOSUB610:IFSS$="Q"THENX$="Q":RETURN
1620  IFSS$=""THEN1610
1630  Q$(QX,GX)=SS$:NC(QX,GX)=0
1640  IFNC(QX,GX)>=MCTHEN1690
1650  CX=NC(QX,GX)+1:PRINTCX;" ";
1660  GOSUB610:IFSS$="Q"THEN1690
1670  IFSS$=""THEN1660
1680  C$(CX,QX,GX)=SS$:NC(QX,GX)=CX:GOTO1640
1690  PRINT:PRINT"POIDS:";:GOSUB610:WT(QX,GX)=VAL(SS$):RETURN
```

* La version Apple de cette ligne en diffère seulement par le nombre d'espaces dans les constantes littérales.

Figure 7.47a : BASIC Réel pour Entréequest

```
# « Editer » les noms des groupes

editnoms  FOR GX = 1 TO 2
              PRINT GX ; " "; GNS(GX)        # afficher nom du groupe
              GOSUB entreechaine             # saisir modif
              IF SS$ <> " " THEN             # RETURN = pas de modif
                  GN$(GX) = SS$
              NEXT GX
          RETURN

1700  FORGX=1TO2:PRINTGX;" ";GN$(GX)
1710  GOSUB610:IFSS$<>""THENGN$(GX)=SS$
1720  NEXTGX:RETURN
```

La routine « editnoms » affiche le nom du groupe pour chacun des deux groupes, et permet au joueur d'entrer un changement de nom. Un RETURN veut dire « pas de changement » ; dans le cas contraire le nom est remplacé par la chaîne de caractères frappée.

Figure 7.48 : Edition — Mise à Jour des Noms de Groupes

Routines d'aiguillage vers « l'édition » des questions

```
editun      GX = 1 : GOSUB edit
            RETURN

editdeux    GX = 2 : GOSUB edit
            RETURN

edit        repeat{
                GOSUB effacecran                          # démarrer sur écran pro-
                                                          pre
                PRINT " * ";                              # signal de saisie
                GOSUB uncar : PRINTX$                     # obtenir une commande
                IF case
                    X$ = " E " THEN GOSUB modquest        # modifier une question
                    X$ = " D " THEN GOSUB annulquest      # annuler une question
                    X$ = " A " THEN GOSUB addquest        # ajouter une question (à
                                                          la fin)
                    X$ = " B " THEN GOSUB insertquest     # insérer une question
                    X$ = " Q " THEN break                 # terminé pour « l'édi-
                                                          tion »
                }
            RETURN
```

```
1740  GX=1:GOSUB1760:RETURN
1750  GX=2:GOSUB1760:RETURN
1760  GOSUB2200:PRINT"*";:GOSUB2180:PRINTX$:IFX$="Q"THENRETURN
1770  IFX$="E"THENGOSUB1990:GOTO1760
1780  IFX$="D"THENGOSUB1850:GOTO1760
1790  IFX$="A"THENGOSUB1820:GOTO1760
1800  IFX$="B"THENGOSUB1910:GOTO1760
1810  GOTO1760
```

Voici le programme commun aux deux commandes « d'édition » 1 et 2. Dans les deux cas, un " * " est affiché comme signal. Les commandes peuvent alors être entrées pour modifier, annuler, ajouter ou insérer une question. Ce sont là les fonctions fondamentales de mise à jour dont on doit disposer chaque fois que l'on a un texte à faire évoluer.

Figure 7.49 : Edition — Mise à Jour des Questions

Vous savez, selon toute vraisemblance, que si b est un nombre entier quelconque, tout autre nombre entier N peut être écrit sous un développement unique d'après la formule :

$$N = a_0 + a_1 \times b + a_2 \times b^2 + ... + a_k \times b^k$$

où a_0, a_1, ..., a_k sont des nombres entiers allant de 0 à b − 1. Par exemple si b = 10,

$$1980 = 0 + 8 \times 10 + 9 \times 100 + 1 \times 1000;$$

ou encore

$$1980 = 0 + 8 \times 10 + 9 \times 10^2 + 1 \times 10^3$$

Il n'y a aucune autre décomposition de 1980 en unités, dizaines, centaines et milles en utilisant des coefficients (dans le cas précis 1, 9, 8 et 0) compris entre 0 et 9.

Pour illustrer la compression des préférences, supposons que chaque question donne lieu au choix entre 9 réponses possibles. Autrement dit, MC vaudra 9. Ainsi les préférences « comprimées »

```
# Ajouter une question au groupe GX
addquest  IF NQ(GX) > = MQ THEN {
               PRINT " PLUS DE PLACE "
               RETURN
               }
          QX = NQ(GX) + 1          # indice de la nouvelle question
          GOSUB entreequest
          IF X$ < > " Q " THEN     # Q interrompt le processus
               NQ(GX) = QX         # mise à jour du nombre de questions
          RETURN

1820  IFNQ(GX) > =MQTHENPRINT"PLUS DE PLACE":RETURN
1830  QX=NQ(GX)+1:GOSUB1600:IFX$ < >"Q"THENNQ(GX)=QX
1840  RETURN
```

Cette routine ajoute une question en fin de celles concernant le groupe GX. L'erreur commise dans la routine « stockernouveau » (fig. 7.22) n'est pas répétée ici. De ce fait, le processus est interrompu immédiatement s'il n'y a pas de place.

Notons l'utilisation de la variable QX et de l'élément de table NQ(GX). QX est immédiatement portée à sa nouvelle valeur mais si la saisie d'une question est interrompue (« entrerquest » retourne XS ="Q")NQ(GX) n'est alors pas modifié.

Figure 7.50 : Addition d'une Question

```
# Supprimer une question du groupe GX

annulquest  GOSUB numquest        # obtenir numéro de question QQ
            IF QQ <> 0 THEN       # QQ = 0 interrompt
                GOSUB sortirquest
            RETURN

1850  GOSUB2160:IFQQ<>0THENGOSUB1870
1860  RETURN
```

Cette routine annule une question du groupe GX. La partie technique de cette suppression se trouve dans la routine « sortirquest ».

Figure 7.51 : Suppression d'une Question

```
# Annuler la question numéro QQ du groupe GX

sortirquest  NQ(GX) = NQ(GX) - 1              # une question en moins
             IF QQ > NQ(GX) THEN              # il n'y a rien à faire
                 RETURN                       # si l'annulation porte sur la
                                              dernière
             FOR QX = QQ TO NQ(GX)            # dans le cas contraire, déca-
                                              ler chaque
                 Q$(QX, GX) = Q$(QX + 1, GX)  # question vers le haut d'un
                                              cran
                 IF NC(QX + 1, GX) > 0 THEN   # décaler les choix, s'il y en a
                     FOR CX = 1 TO NC(QX + 1, GX)
                         C$(CX, QX, GX) = C$(CX, QX + 1, GX)
                         NEXT CX
                 NC(QX, GX) = NC(QX + 1, GX)  # décaler le nombre de choix
                 WT(QX, GX) = WT(QX + 1, GX)  # décaler le poids
                 NEXT QX
             RETURN

1870  NQ(GX)=NQ(GX)-1:IFQQ>NQ(GX)THENRETURN
1880  FORQX=QQTONQ(GX):Q$(QX,GX)=Q$(QX+1,GX):IFNC(QX+1,GX)<=
      0THEN1900
1890  FORCX=1TONC(QX+1,GX):C$(CX,QX,GX)=C$(CX,QX+1,GX):NEXTCX
1900  NC(QX,GX)=NC(QX+1,GX):WT(QX,GX)=WT(QX+1,GX):NEXTQX:RETURN
```

La routine « sortirquest » assure la technique de suppression de la question numéro QQ du groupe GX. Le libellé de la question, les choix, le nombre des choix et le poids doivent tous être décalés, au fur et à mesure que chaque question située derrière la question supprimée, est remontée d'un cran dans la liste. Le cas de la question sans choix est prévu, bien qu'il ne doive pas se produire.

Figure 7.52 : Technique de Suppression d'une Question

```
# Insérer une question dans le groupe GX

insertquest  IF NQ(GX) > = MQ THEN {                 # si pas de place
                  PRINT " PLUS DE PLACE "            # le dire et revenir
                  RETURN
                  }
             PRINT " JUSTE AVANT ";                  # trouver où l'on peut bien
                                                     mettre cette question
             GOSUB numquest
             IF QQ = 0 THEN RETURN                   # QQ = 0, si le joueur inter-
                                                     rompt le processus
             FOR QX = NQ(GX) TO QQ STEP − 1          # décaler d'abord la dernière
                                                     question
                  Q$(QX + 1, GX) = Q$(QX, GX)
                  IF NC(QX, GX) > 0 THEN             # décaler les choix, s'il y en a
                       FOR CX = 1 TO NC(QX, GX)
                            C$(CX, QX + 1, GX) = C$(CX, QX, GX)
                            NEXT CX
                  NC(QX + 1, GX) = NC(QX, GX)        # décaler le nombre de choix
                  WT(QX + 1, GX) = WT(QX, GX)        # décaler le poids
                  NEXT QX
             NQ(GX) = NQ(GX) + 1                     # mettre à jour le nombre de
                                                     questions
             QX = QQ : GOSUB entreequest             # saisir la question
             IF X$ = " Q " THEN                      # Q pour interrompre
             GOSUB sortirquest                       # restituer la place
             RETURN
```

```
1910  IFNQ(GX)>=MQTHENPRINT"PLUS DE PLACE":RETURN
1920  PRINT"JUSTE  AVANT";:GOSUB2160:IFQQ=0THENRETURN
1930  FORQX=NQ(GX)TOQQSTEP−1:Q$(QX+1,GX)=Q$(QX,GX)
1940  IFNC(QX,GX)<=0THEN1960
1950  FORCX=1TONC(QX,GX):C$(CX,QX+1,GX)=C$(CX,QX,GX):NEXTCX
1960  NC(QX+1,GX)=NC(QX,GX):WT(QX+1,GX)=WT(QX,GX):NEXTQX
1970  NQ(GX)=NQ(GX)+1:QX=QQ:GOSUB1600:FX$="Q"THENGOSUB1870
1980  RETURN
```

La routine « insertquest » insère une question dans les questions du groupe GX. Le joueur est prié de préciser le numéro de la question avant laquelle la nouvelle question doit être insérée. La place pour la nouvelle question est libérée en décalant chaque question (à partir de celle dont le numéro a été spécifié) vers la position immédiatement supérieure. Pour l'insertion, il est essentiel que chaque question soit d'abord décalée vers son nouvel emplacement avant de considérer comme disponible (pour la question de rang immédiatement inférieur) son ancien emplacement.

Figure 7.53 : Insertion d'une Question

seront représentées par 1 nombre en base 10 (MC + 1). Chaque chiffre entre zéro et neuf représente le numéro de la réponse choisie qui est associé à l'une des valeurs de préférence (zéro signifiant qu'aucune des réponses choisies, n'a été affectée à la valeur de préférence). A la figure 7.13, les préférences de SUSAN pour la question un, seraient encodées en 5432 et celles pour les réponses de la question deux, donneraient le nombre 1203.

Lorsqu'il y a peu de réponses à choisir par question et peu de valeurs de préférence, ce procédé permet de mettre dans un seul nombre, les préférences du joueur pour une question. Cela fonctionne jusqu'au moment où le nombre est si grand qu'il est « arrondi ». La limite à laquelle cet « arrondi » survient, varie avec les systèmes. Sur Apple ou Pet, c'est neuf chiffres ; sur TRS-80 c'est six chiffres, exception faite de l'utilisation d'une variable « double précision ». Dans ce cas, le tableau de dimensions 6 × 12 × 2, qui prenait 1 104 octets, en prendrait 2 196. La valeur maximum du nombre qui code les préférences, peut être obtenue par la formule suivante :

M = (MC + 1)↑NF − 1

Par exemple, si MC a une valeur de 9 et NF (nombre des valeurs de préférence), est égal à 4, M sera alors égale à 9999.

Nous venons de voir comment le codage des préférences peut économiser beaucoup de mémoire. Le prix à payer pour cette économie apparaît dans le sous-programme *jumelage* (fig. 7.33) où ont lieu 2* NP(GC) appels à la routine *score* (fig. 7.34) ; la routine *score* appelle à son tour la routine *correspondance* (fig. 7.35), ce qui se traduit par

NP(GC) * NQ(GC) + NP(GC) * NQ(GP)

appels à la routine *expansepref*. La totalité de ces appels pourrait être éliminée si les grands tableaux dont nous avons parlé précédemment, étaient utilisés. Mais beaucoup de petits systèmes ne pourraient pas réserver la place en mémoire nécessaire à un tel tableau. Cependant on peut s'arrêter à un compromis intermédiaire. Dans la première partie du sous-programme *jumelage*, les préférences exprimées par le joueur sont confrontées aux réponses de chacun des joueurs de l'autre groupe, de sorte que les préférences codées du joueur sont décomprimées NP(GC) fois. Mais si ces préférences étaient décompressées une seule fois et sauvegardées dans une version à deux dimensions du tableau FT, le premier terme de la somme indiquée auparavant, pourrait être réduit de NP(GC) * NQ(GC) à NQ(GC). Le tableau FT à deux dimensions aurait pour celles-ci les valeurs de NF et de MQ. En reprenant les

valeurs que nous avons déjà utilisées, cela signifierait que le tableau FT serait augmenté de **4** à 4 × 6 = **24** éléments et que le nombre d'appels à *expansepref* serait diminué de 12 × 6 + 12 × 6 soit **144** à 6 + 12 × 6 soit **78**. Il semble que ce soit un encombrement mémoire mineur pour un gain de performance faible. Mais si le nombre de joueurs par groupe passait de 12 à 24, il n'y aurait pas d'encombrement mémoire supplémentaire, et le nombre des appels de sous-programme diminuerait de 24 × 6 + 24 × 6 soit **288** à 6 + 24 × 6 soit **150**.

```
# Modifier une question du groupe GX
modquest  GOSUB munquest                        # numéro de question
          IF QQ = 0 THEN
               RETURN
          PRINT Q$(QQ, GX)                      # afficher la question
          GOSUB entreechaine                    # saisir la modification
          IF SS$ = "Q" THEN                     # Q = pas de modif. et
                                                abandon
               RETURN
          else if SS$ <> "" THEN                # RETURN = pas de modi-
                                                fication
               Q$(QQ, GX) = SS$                 # enregistrer nouvelle chaî-
                                                ne
          GOSUB choix                           # modifier les choix
          PRINT : PRINT "POIDS :" ; WT(QQ, GX) ;   # afficher poids
                                        ":" ;
          GOSUB entreechaine                    # saisir la modification
          IF VAL(SS$) <> 0 THEN                 # non-numérique ou nul
               WT(QQ, GX) = VAL(SS$)            # veut dire : pas de modi-
                                                fication
          RETURN
```

```
1990  GOSUB2160:IFQQ=0THENRETURN
2000  PRINTQ$(QQ,GX):GOSUB610:IFSS$="Q"THENRETURN
2010  IFSS$<>""THENQ$(QQ,GX)=SS$
2020  GOSUB2050:PRINT:PRINT"POIDS:";WT(QQ,GX);":";:GOSUB610
2030  IFVAL(SS$)<>0THENWT(QQ,GX)=VAL(SS$)
2040  RETURN
```

La routine « modquest » permet au joueur de modifier une question du groupe GX. La modification des choix (réponses possibles) est réalisée par la routine « repchoix ».

Figure 7.54 : Edition — Mise à Jour d'une Question

```
# Modifier les choix (réponses) de la question (QQ, GX)

choix   CX = 1                                          # initialiser indice du
                                                        choix
        while (CX < = NC(QQ, GX)) {                     # pour chaque choix en
                                                        cours
            PRINT CX ; "" ; C$(CX, QQ, GX)              # afficher numéro et libellé
            GOSUB entreechaıne                          # et saisir la modification
            IF SS$ = " Q " THEN RETURN                  # Q = abandon
            else IF SS$ = " D " THEN {                  # D = supprimer le choix
                NC(QQ, GX) = NC(QQ, GX) - 1             # un choix de moins
                IF CX < = NC(QQ, GX) THEN               # si ce n'est pas le dernier
                                                        choix qu'on supprime
                    FOR CY = CX TO NC(QQ, GX)           # décaler vers le haut les
                                                        choix suivants
                        C$(CY, QQ, GX) = C$(CY + 1, QQ, GX)
                        NEXT CY
                }
            else {                                      # non-Q ou D
                IF SS$ < > "" THEN                      # si chaîne non vide
                    C$(CX, QQ, GX) = SS$                # remplacer choix par SS$
                CX = CX + 1
                }
            }
        while (NC(QQ, GX) < MC) {                       # tant qu'il y a de la place
            CX = NC(QQ, GX) + 1                         # incrementer indice de
                                                        choix et
            PRINT CX ; "" ; : GOSUB entreechaıne        # demander un choix
            IF SS$ ="" or " Q" THEN RETURN              # RETURN ou Q met fin à
                                                        l'opération
            else {
                C$(CX, QQ, GX) = SS$                    # choix = SS$
                NC(QQ, GX) = CX                         # le compter
                }
            }
        RETURN
```

La routine « choix » permet au joueur de modifier les choix (réponses possibles) correspondant à la réponse dont les indices sont (QQ, GX). Tout d'abord les choix en cours sont appelés et le joueur peut modifier ou annuler chacun d'eux. (D = annuler, RETURN = pas de changement.) Puis on peut ajouter de nouveaux choix en fin de liste. Dans cette phase d'addition la sortie du processus se fait par Q ou RETURN.

Au début du programme, CX est contrôlé de façon explicite. Une boucle FOR...NEXT ne pourrait y être utilisée, car la suppression du choix peut modifier la limite supérieure NC(QQ,GX).

Figure 7.55 : Edition — Mise à Jour des Réponses Choisies

Nous voici à la fin de notre étude sur l'économie de place mémoire. (Vous pourriez avoir envie de réaliser la modification dont nous venons de parler.) De nombreux points intéressants sont soulevés par les problèmes de l'optimisation des liaisons entre *jumelage*, *score*, *correspondance* et *expansepref* dans cette nouvelle organisation. Par exemple, doit-on récrire *expansepref* de façon à décompresser toutes les préférences du joueur tout de suite ? Sinon, faudra-t-il deux versions de FT, une à une seule, l'autre à deux dimensions ? Comment mesurerez-vous le gain de temps résultant de ces changements ?

```
2050  CX=1
2060  IFCX>NC(QQ,GX)THEN2130
2070  PRINTCX;"";C$(CX,QQ,GX):GOSUB610:IFSS$="Q"THENRETURN
2080  IFSS$<>"D"THEN2110
2090  NC(QQ,GX)=NC(QQ,GX)-1:IFCX>NC(QQ,GX)THEN2060
2100  FORCY=CXTONC(QQ,GX):C$(CY,QQ,GX)=C$(CY+1,QQ,GX):NEXTCY:GOTO2060
2110  IFSS$<>""THENC$(CX,QQ,GX)=SS$
2120  CX=CX+1:GOTO2060
2130  IFNC(QQ,GX)=>MCTHENRETURN
2140  CX=NC(QQ,GX)+1:PRINTCX;"  ";:GOSUB610;IFSSS=""ORSSS="Q"THENRETURN
2150  C$(CX,QQ,GX)=SS$:NC(QQ,GX)=CX:GOTO2130
```

Figure 7.55a : BASIC Réel pour « choix »

```
# Demander le numéro d'une question du groupe GX ; QQ sera = 0 si non valide

numquest  PRINT "NUMERO DE QUESTION : ";
          GOSUB entreechaine
          QQ = VAL(SS$)
          IF QQ < 0 OR QQ > NQ(GX) THEN
               QQ = 0
          RETURN

2160  PRINT"NUMERO DE QUESTION: ";:GOSUB610:QQ=VAL(SS$):
        IFQQ<0ORQQ>NQ(GX)THENQQ=0
2170  RETURN
```

La routine « numquest » demande un numéro de question et contrôle sa validité.

Figure 7.56 : Entrée du Numéro de Question

Ceci met fin à notre analyse du programme du Jumelage. Lisez les listings des programmes, en vous rappelant ce que nous avons étudié et essayez de trouver ce que vous pouvez en apprendre de plus.

Modifications et améliorations

Après notre étude du programme du Jumelage (ou au cours de celle-ci), il se peut que vous désiriez réaliser quelques-unes des modifications ou améliorations suivantes :

- Réaliser le stockage externe selon les indications existant dans le programme.
- Si ce stockage externe est réalisé, ou si la phase préparatoire est réamorcée, alors qu'on vient d'achever la saisie des fichiers « joueur », les données « joueur » et les données « questions » peuvent devenir incompatibles.
  Envisagez les possibilités suivantes :
  - Modification automatique des fichiers « joueur » en cas d'addition ou de suppressions de questions.
  - Réencodage automatique des préférences du joueur, si la valeur de MC ou celle de NF vient à changer.
  - Automatiquement, ou sur entrée de commande, effacer les fichiers « joueur » dans certains cas.

```
# Quelques vieilles connaissances (en versions Pet)
uncar         repeat
                   GET X$
                   until (X$ <>"")
              RETURN
effacecran    PRINT "clr";
              RETURN

   2180  GETX$:IFX$=""THEN2180
   2190  RETURN
   2200  PRINTCHR$(147);:RETURN
```

Les versions Apple et TRS-80 de ces routines sont présentées en figures 2.11 et 2.12.

Figure 7.57 : Entrée Mono-Caractère et Effacement Ecran

Réaliser tel ou tel point qui vous semble avantageux. Quels sont les problèmes que ces modifications ne résoudront pas ?

- Mettre en place des commandes d'édition du fichier « joueur » qui vous permettent de préciser le point du cycle de mise à jour, d'où vous désirez partir.
- Mettre en place des commandes mono-caractères, qui vous permettent de passer, lors de la mise à jour, à la préférence ou à la question suivante, sans avoir à utiliser continuellement la touche RETURN pour épuiser la liste des valeurs de préférence que vous êtes en train de traiter.
- Remédier au problème propre à *stockernouveau* (fig. 7.22).
- Réaliser une fonction de « jumelage » de « masse » qui génère une liste de couples d'une façon telle qu'elle maximise le total de tous les scores préférentiels des joueurs ayant un partenaire. Si l'un des groupes est plus nombreux que l'autre, assigner à certains membres du groupe de moindre importance, du moins un membre du groupe le plus grand.

```
# Routines de stockage externe

lecture          GOSUB stockextern
                 RETURN

ecriture         GOSUB stockextern
                 RETURN

chargerjoueurs   GOSUB stockextern
                 RETURN

sauverjoueurs    GOSUB stockextern
                 RETURN

stockextern      PRINT "STOCKAGE EXTERNE NON REALISE"
                 RETURN

2210  GOSUB2250:RETURN
2220  GOSUB2250:RETURN
2230  GOSUB2250:RETURN
2240  GOSUB2250:RETURN
2250  PRINT"NON REALISE":RETURN
```

Ces routines sont des « souches » qui indiquent à quelles adresses dans les programmes seront écrites le cas échéant les instructions nécessaires au chargement et à la sauvegarde des données sur mémoire externe.

Figure 7.58 : Stockage sur Mémoire Externe

```
# Initialisation

init  GM = 2                                  #deux groupes autorisés
      READ MQ, MC, MP                         # max. de questions, choix, joueurs
      DATA maxquestions, maxchoix, maxjoueurs
      DIM NQ(GM), NP(GM),                     # numéros des questions et des joueurs ;
      GN$(GM), NM$(MP, GM)                    # noms des groupes,
                                              # nom des joueurs
      Q$(MQ, GM), WT(MQ, GM)                  # questions, poids,
      NC(MQ, GM), C$(MC, MQ, GM),             # numéros des choix, choix
      A(MQ, MP, GM), W(MQ, MP, GM)            # réponses des joueurs et préférences
      READ MT : DATA maxscores                # taille du tableau « scores maxi »
      DIM TS(MT), TI(MT)                      # scores maxi et numéros de joueurs
      READ NF : DATA nconcord                 # taille des tableaux de préférences
      DIM FV(NF), FT(NF)                      # tableaux des valeurs de préférence
                                              et des valeurs de préférence étendues
      FOR FX = 1 TO NF                        # lire les valeurs de préférence
            READ FV(FX)
            NEXT FX
      DATA plus grande valeur de préférence, ..., plus petite
      RETURN

2260* GM=2:READMQ,MC,MP:DATA2,5,2
2270  DIMNQ(GM),NP(GM),GN$(GM),NM$(MP,GM),Q$(MQ,GM)
2280  DIMWT(MQ,GM),NC(MQ,GM),C$(MC,MQ,GM)
2290  DIMA(MQ,MP,GM),W(MQ,MP,GM)
2300  READMT:DATA3
2310  DIMTS(MT),TI(MT)
2320  READNF:DATA4
2330  DIMFV(NF),FT(NF)
2340  FORFX=1TONF:READFV(FX):NEXTFX
2350  DATA2,1,-1,-2
2360  RETURN
```

Les instructions DATA permettent de fixer les dimensions des tableaux. Pour un petit ordinateur individuel (8K de mémoire vive) les tailles de tableaux doivent être très faibles (par exemple 2 questions avec 5 choix et 2 joueurs par groupe). Des valeurs nettement plus grandes peuvent être utilisées sur de plus grands systèmes.

---

* Les valeurs de « maxquestions », « maxchoix » et « maxjoueurs » indiquées sur cette ligne sont suffisamment faibles pour que le programme puisse tourner sur un Pet de 8K. De bien plus grandes valeurs sont possibles sur les systèmes plus importants. Sur un TRS-80 de 16K, par exemple, des valeurs de 10, 5 et 30 peuvent être utilisées. Rappelez-vous que « maxchoix » ne doit pas dépasser 9.

Figure 7.59 : Initialisation

En résumé

Le jeu du Jumelage utilise des séries de questions à réponses multiples pour former des couples entre les membres de deux groupes. Le programme correspondant illustre des principes généraux de conception, et des techniques générales et particulières de programmation. Pour être plus précis, les principes généraux de conception étudiés sont la modularité, la démarche descendante, la localisation des fonctions, l'anticipation des modifications et l'utilisation de souches pour bâtir la structure. Les techniques générales de programmation décrites, comprennent la conception des documentations « utilisateur » et « programmeur », celle des interactions homme-machine et le traitement des erreurs. Les techniques particulières de programmation concernent la conception des fonctions de mise à jour, les algorithmes d'insertion et de suppression, la conception des fonctions de gestion de fichiers et les méthodes permettant d'économiser la place en mémoire.

CHAPITRE 8

Craps

CRAPS est une adaptation sur ordinateur individuel de ce jeu très populaire dans les casinos. L'ambiance animée des casinos a pu être jusqu'à un certain point reconstituée en utilisant certaines techniques simples de programmation et de conception.

Les règles de Craps

Dans sa forme de base, Craps est un jeu de dés simple. Lorsque vient leur tour, les joueurs prennent les dés et les jettent à plusieurs reprises. Ainsi, quand votre tour arrive, vous commencez par parier une certaine somme. Si les jets effectués amènent un coup gagnant, vous ramassez alors votre gain, faites un autre pari et recommencez à jeter les dés. Si les jets amènent un coup perdant, vous payez alors le montant parié et passez les dés au joueur suivant.

Si au début d'une série, vous jetez 7 ou 11 (il s'agit de la somme des points figurant sur les faces des deux dés), vous gagnez tout de suite. Si votre premier coup est 2, 3 ou 12, vous perdez, par contre, immédiatement. Si vous tirez l'une des six autres possibilités (4, 5, 6, 8, 9, 10), ce nombre tiré devient votre « point ». Vous devez continuer à jeter les dés jusqu'à ce que vous tiriez 7, ce qui vous fait perdre, ou que vous tiriez votre point, ce qui vous fait gagner. C'est ainsi que si votre point est 8, vous continuez à jeter les dés, en espérant le 8, jusqu'à ce qu'il sorte et gagne, ou bien que 7 tombe, ce qui vous fait perdre. Tous les autres totaux que vous obtenez entre-temps, y compris 2, 3, 11 et 12, sont ignorés. Par exemple, la séquence suivante est gagnante :

8, 5, 11, 6, 9, 3, 8

C'est cette forme fondamentale de Craps qui a été réalisée ici. A une table réelle de craps dans un casino, la taille du tapis rend possible les paris au-delà et au-dessus du pari principal fait par le joueur. Par exemple, vous pouvez parier que le prochain coup donnera 3. Pour ce faire, vous placez votre jeton à l'endroit adéquat du tapis. Si le jet suivant ne donne pas 3, le croupier ramasse votre jeton ; si au contraire c'est 3, on vous paye 12 jetons en les posant à côté du vôtre. Si vous ne voulez pas parier ces 13 jetons sur la sortie de 3 au coup suivant, vous devez alors les ramasser avec célérité — l'action étant plutôt rapide à une table de craps et le nombre des enjeux tel, qu'il est facile d'oublier où l'on a placé ses jetons.

GAGNE ENJEU CETTE SERIE MEILLEURE SERIE
0 0 0 0

Situation initiale.

Le joueur tape N.

POURRAIS-JE AVOIR VOTRE NOM
S'IL VOUS PLAIT ? **JOHN**
MERCI.

Le programme demande le nom et le joueur le fournit.

GAGNE ENJEU CETTE SERIE MEILLEURE SERIE
0 0 0 0
JOHN

Situation initiale (nom indiqué).

Le joueur a répondu à l'affichage de la situation initiale en tapant N (commande du « nouveau joueur »). Le programme réclame le nom, et le joueur répond JOHN. Le programme dit MERCI. Alors, après une pause de faible durée, il réaffiche la situation initiale qui comporte maintenant le nom du joueur.

Figure 8.1 : JOHN Joue le Premier

Ces possibilités de pari supplémentaires n'ont pas été incorporées dans la présente version du jeu parce que la représentation de la table sur l'écran aurait été difficile et aussi, parce qu'il n'y a pas de moyens d'entrée et de sortie des données affectés en propre à chaque joueur. Néanmoins, des possibilités de pari supplémentaires pourraient être introduites. Ces possibilités font partie des améliorations suggérées en fin de chapitre.

Le programme de Craps commence par afficher l'écran de début de jeu (fig. 8.1). On peut alors entrer plusieurs commandes. Le jeu peut se jouer à un seul joueur ou à plusieurs. S'il y a plusieurs joueurs, la commande N peut être utilisée pour entrer le nom de chacun (voir figure 8.1). La figure 8.2 montre comment le programme dit à chaque joueur où se situe son tour dans le déroulement du jeu. (*Nota* : l'entrée des noms se faisant par INPUT, les noms en deux mots (par exemple JOHN SMITH) doivent être placés entre guillemets.)

Supposons qu'on ait ajouté MARY et SUSAN et qu'il y ait maintenant trois joueurs : c'est toujours le tour de JOHN qui doit maintenant parier. On le voit à la figure 8.3. Après avoir entré une somme de 25, JOHN commence à jeter les dés en appuyant sur la barre d'espacement (une seule fois).

La figure 8.4 montre ce qui arrive ensuite. Tout d'abord l'image d'une paire de dés apparaît. Puis le résultat du coup est annoncé en toutes lettres (par exemple, DESOLE, C'EST PERDU ou bien LE POINT EST 5). Après une pause brève, l'image des dés disparaît, mais le résultat reste affiché. Si ce résultat est un gain, la situation initiale de JOHN réapparaît alors ; si c'est une perte (voir figure 8.4), on visualise la situation initiale du joueur suivant (ici, c'est MARY).

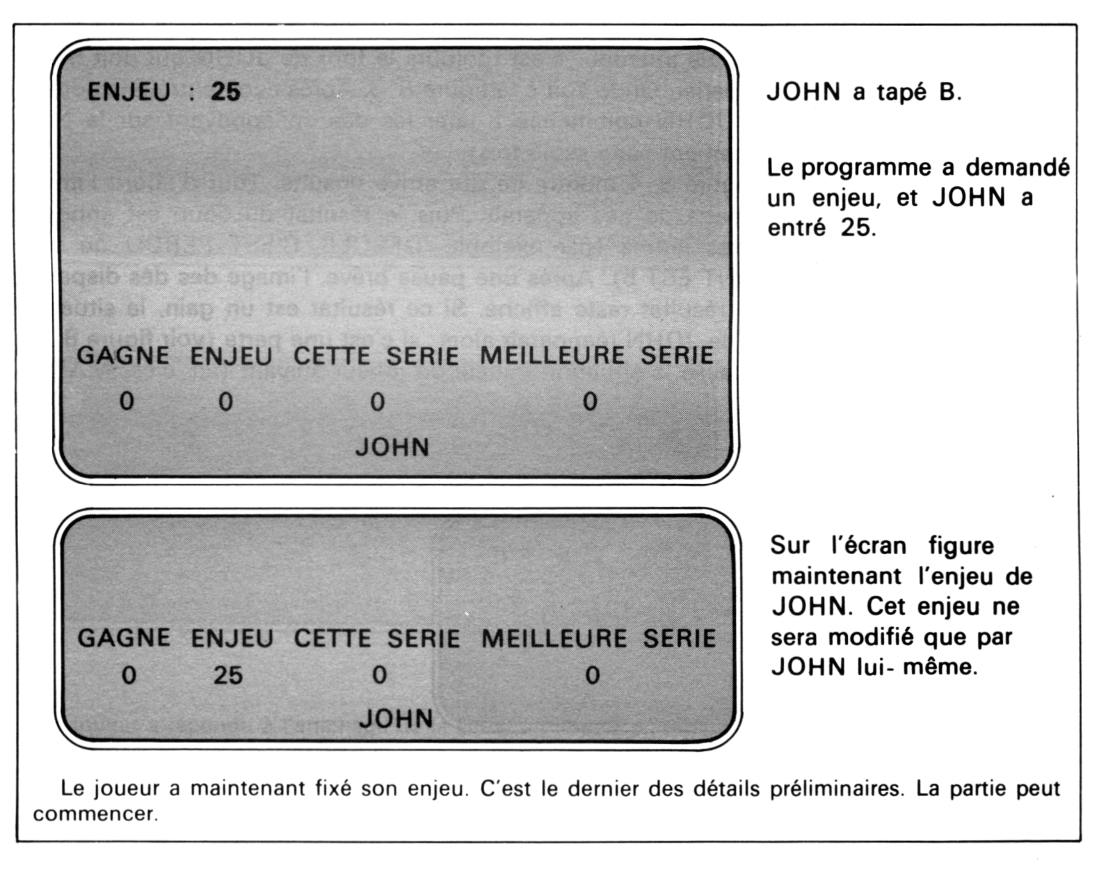
```
POURRAIS-JE AVOIR VOTRE NOM S'IL
VOUS PLAIT ? MARY
MERCI. VOUS JOUEREZ APRES JOHN.
```

En réponse à une autre commande N, le programme a demandé le nom du joueur. Le joueur en question a entré au clavier le nom de MARY et le programme a placé MARY après JOHN dans l'ordre de jeu.

Figure 8.2 : MARY Jouera après JOHN

Si le coup n'est ni une perte ni un gain immédiat, une ligne est affichée pour indiquer la valeur du point et le nombre de lancements.

C'est ce dernier cas qui se produit pour MARY (fig. 8.5). MARY entre un enjeu de 50 et appuie sur la barre d'espacement pour jeter les dés. Un 6 et un 2 apparaissent, puis le résultat (LE POINT EST 8) et finalement « la ligne de la partie en cours » (POINT : 8 LANCEMENTS : 1). Les dés disparaissent et le programme attend que l'on jette les dés. MARY attend l'instant magique, appuie d'une manière personnelle sur la barre d'espacement et deux 4 apparaissent — MARY a gagné. La situation initiale de MARY reparaît. Les GAINS, CETTE SERIE, ainsi que la MEILLEURE SERIE ont été mises à jour, tandis que la valeur de l'ENJEU n'a pas changé.



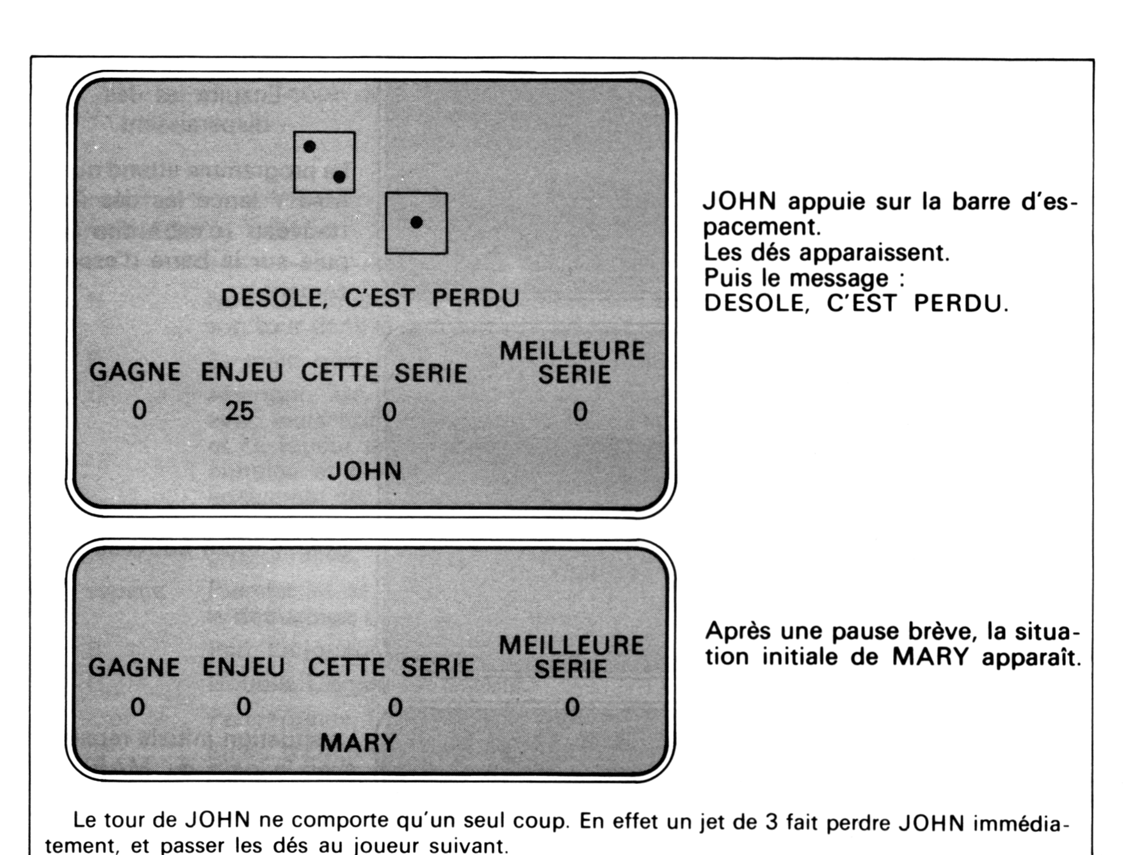
Le joueur a maintenant fixé son enjeu. C'est le dernier des détails préliminaires. La partie peut commencer.

Figure 8.3 : La Somme Mise en Jeu par JOHN

Maintenant que vous avez une idée générale du jeu, voyons la liste des commandes qu'il est possible d'entrer au moment de l'affichage de la situation initiale. Cette liste est donnée en figure 8.6. Nous allons maintenant commenter ces commandes l'une après l'autre.

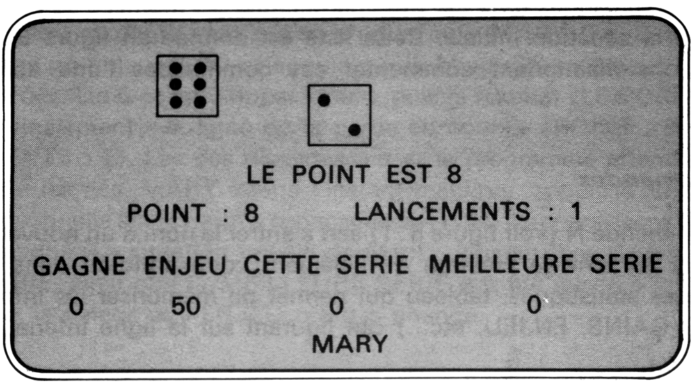
Les commandes

La commande N (voir figure 8.1) sert à entrer le nom d'un nouveau joueur. N entraîne la création de l'élément correspondant dans le tableau des statistiques, tableau qui permet de mémoriser les informations (GAINS, ENJEU, etc...) qui figurent sur la ligne inférieure



Le tour de JOHN ne comporte qu'un seul coup. En effet un jet de 3 fait perdre JOHN immédiatement, et passer les dés au joueur suivant.

Figure 8.4 : JOHN Lance les Dés

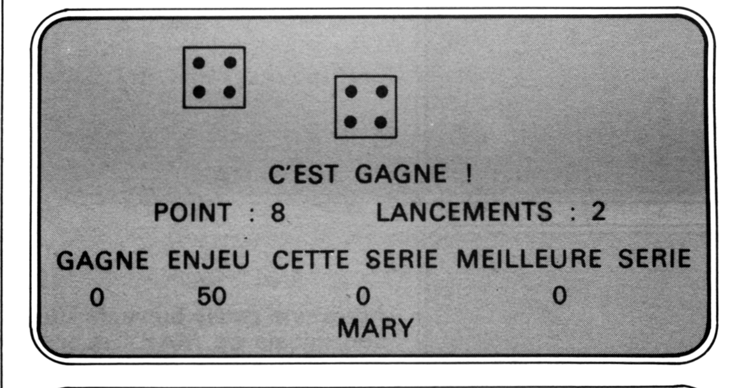


L'image des dés apparaît en premier.
Puis « LE POINT EST 8 »
Puis « POINT : 8 COUPS:1 »

LE POINT EST 8
POINT : 8 LANCEMENTS : 1
GAGNE ENJEU CETTE SERIE MEILLEURE SERIE
0 50 0 0
MARY

Ensuite les dés disparaissent.

Le programme attend que MARY lance les dés à nouveau (c'est à-dire appuie sur la barre d'espacement.)



MARY tire à nouveau 8.

GAGNE ENJEU CETTE SERIE MEILLEURE SERIE
50 50 1 1
MARY

La situation initiale reparaît avec le gain de MARY.

Figure 8.5 : MARY Réussit son Point

de votre situation initiale pendant que les autres personnes sont en train de jouer. Ainsi, quand reviendra votre tour, le programme en saura autant sur vous qu'il en savait lorsque votre tour précédent s'est terminé.

La commande P vous permet de passer les dés au joueur suivant sans attendre d'avoir perdu. (Autrement vous garderez la main aussi longtemps que vous gagnez.) Vos données sont enregistrées et celles du joueur suivant sont rappelées.

La commande Q permet à un joueur (n'importe lequel — pas seulement celui dont la situation initiale figure sur l'écran) de quitter la partie. La figure 8.7 montre le dialogue correspondant. La liste des noms et numéros de joueurs est affichée. Tout joueur peut alors quitter la partie, en entrant tout simplement son propre numéro.

La commande S n'a aucun effet. (Quand nous parlerons du programme, nous expliquerons pourquoi nous l'avons prévue.)

La barre d'espacement sert à faire démarrer la partie. Le premier coup de dés apparaît sur l'écran et la partie continue comme il a été

Commande	**Effet**
N	Nouveau joueur. Le programme demande son nom et indique quel sera son tour dans la partie.
P	La main passe. La situation initiale du joueur suivant apparaît.
Q	Abandon. Un joueur est autorisé à quitter la partie. La liste des joueurs avec leurs numéros est affichée. Le numéro du joueur sortant est entré, et ce joueur est retiré de la partie (si l'on frappe zéro la commande est annulée et personne ne sort). Les gains et pertes des joueurs figurent également sur la liste.
S	Paris complémentaires. Il ne se passe rien, cette possibilité n'ayant pas été programmée.
espace	Premier jet de dés. Le premier coup du joueur est effectué, ce qui assure le démarrage de la série en cours.
B	Pari. Le joueur peut entrer un enjeu sous la forme d'un nombre.
D	Double. L'enjeu est doublé.
L or W	Pertes/gains. Leur montant total devient l'enjeu.
O	Enjeu d'Origine. Le dernier montant entré par une commande B, devient l'enjeu du pari.
R	Laisser courir. L'enjeu devient égal au dernier enjeu plus le montant gagné. Cette commande a le même effet que celui de la commande D.

Figure 8.6 : Commandes Disponibles à l'Affichage de la Situation Initiale

indiqué. Les commandes influant sur le pari, permettent des modifications diverses de la somme mise en jeu. La figure 8.3 montre comment utiliser la commande B pour entrer un enjeu sous forme numérique. Les autres commandes (D, L, O, R, W) modifient la valeur de l'enjeu sur la ligne des statistiques, sans modifier le reste de l'écran.

GAGNE	ENJEU	CETTE SERIE	MEILLEURE SERIE
25	25	0	2

JOHN

Situation initiale de JOHN.

SUSAN désire quitter la partie.
Un joueur tape la lettre Q.

NUMERO	GAGNE/PERDU	NOM
1	25	JOHN
2	– 700	MARY
3	– 20	SUSAN

QUEL EST VOTRE NUMERO, S'IL VOUS PLAIT ? **3**

SUSAN s'en va.

GAGNE	ENJEU	CETTE SERIE	MEILLEURE SERIE
25	25	0	2

JOHN

La situation initiale de JOHN reparaît.

Quelque temps plus tard, c'est à nouveau le tour de JOHN, et SUSAN désire quitter la partie. La commande Q provoque l'apparition d'une liste qui permet d'identifier SUSAN comme ayant le numéro 3. Après le départ de SUSAN, c'est toujours à JOHN de jouer.

Figure 8.7 : On Perd SUSAN

Le programme de Craps

Le programme qui réalise Craps, est présenté de la figure 8.8 à la figure 8.39. De nombreux aspects ont déjà été commentés dans les chapitres précédents et ne le seront plus ici. Il s'agit des points suivants :

— Modularité, structure descendante.

— Localisation des fonctions (par exemple, *jeterdés* et *dessinedés*).

— Identification du joueur par choix dans un menu (fig. 8.31 et 8.32).

— Technique de suppression d'un élément d'un tableau (fig. 8.32).

— Gestion de Fichiers (fig. 8.34).

Nous concentrerons nos commentaires sur plusieurs nouveaux points :

— Utilisation de l'écran pour y présenter un mélange d'informations constantes et d'informations changeantes.

— Rapidité et continuité dans le déroulement du jeu.

— Commutation de contexte.

— Tirage au hasard.

Utilisation de l'écran

Le programme de Craps présenté ici, repose sur l'utilisation d'un écran vidéo. Le caractère même du jeu serait en grande partie perdu, si l'on devait le transformer pour qu'il fonctionne sur un terminal imprimant du genre télétype. Au début de chaque série de coups, l'écran est effacé. Les différents éléments du jeu sont alors affichés à des places séparées sur l'écran : les dés se trouvent à un endroit, les statistiques à un autre. Il en est de même pour le résultat et la « ligne du coup en cours ». Chacun de ces éléments gère l'emplacement qui lui revient sur l'écran, effaçant l'information précédente ou la remplaçant par une information nouvelle lorsque cela est nécessaire.

La gestion indépendante de plusieurs zones sur l'écran repose sur une routine de positionnement du curseur (fig. 8.27), qui permet de placer ce dernier sur une ligne et une colonne déterminées. Les lignes et les colonnes attribuées aux différents éléments à afficher sont précisées dans la routine *initialisation* (fig. 8.35).

L'affichage des dés est assuré par les routines *dessinedés* et *effacedés* (fig. 8.38 et 8.39). La routine *dessinedés* qui est simple

```
# Craps
     GOSUB initialisation                # préciser constantes
     repeat {
          GOSUB depart                   # effacer l'ancienne partie, afficher
                                         statistiques
          GOSUB uncar                    # entrée mono-caractère
          GOSUB codentre                 # donner à IC valeur selon commande
          ON IC GOSUB
               enjeu,                    # fixer montant enjeu
               partie,                   # jouer une partie
               nouveau,                  # un nouveau joueur rentre dans la partie
               abandon,                  # un joueur quitte la partie
               parisecondaire,           # tous enjeux en plus de l'enjeu principal
               passer                    # passer les dés
          }
```

```
100   GOSUB 1950
130*  GOSUB 960:GOSUB 2340:GOSUB 160
140   ON IC GOSUB 560,230,1400,1690,1820,1560
150   GOTO 130
```

Voici la boucle principale de Craps. Chaque itération correspond à une partie. Le joueur commence par entrer un caractère qui détermine l'événement suivant. Un espace démarre la partie et déclenche le jet du premier coup de dés. Les autres commandes servent à faire entrer ou sortir des joueurs, ou parier des enjeux autres que l'enjeu principal du joueur. Le joueur peut passer les dés ou modifier son enjeu. Ces derniers peuvent être entrés numériquement, doublés, fixés au dernier montant des gains totaux ou des pertes ou ramenés à « l'ancien » montant (le dernier enjeu entré sous forme de nombre). On peut également « laisser courir » les gains.

La routine « départ » affiche le nom du joueur en cours et ses statistiques. Dès que le joueur perd, les dés passent au suivant.

* En raison de la longueur de la routine d'initialisation des versions Apple et TRS-80, les routines des figures 8.37, 8.38 et 8.39 ont des numéros de ligne différents dans les versions Apple et TRS-80.

Routine	*Ligne Pet*	*Ligne Apple/TRS-80*	*Figure*
dessinedés	2240	2500	8.38
effacedés	2300	2580	8.39
uncar	2340	2660	8.37

C'est ainsi que le « GOSUB 2340 » de la ligne 130 devient « GOSUB 2660 » dans les versions Apple et TRS-80.

Figure 8.8 : Craps

(fig. 8.38), est basée sur l'utilisation des caractères de déplacement du curseur du Pet. Après avoir positionné le curseur à l'endroit prévu pour chaque dé, *dessinedés* trace la chaîne de caractères BX$, ce qui se traduit par l'apparition d'un carré sur l'écran. Ce carré contiendra les différents « points » du dé. Le curseur est placé dans le coin supérieur gauche de cette zone carrée. La version Pet de la chaîne BX$ contient les caractères curseur gauche, curseur bas et curseur haut, entrelacés avec les caractères qui donnent les lignes horizontales et verticales (shift $, shift #, shift % et shift ')* et réalisent la boîte carrée. Une fois cet encadrement dessiné, on affiche l'une des six faces possibles du dé à partir d'une chaîne tirée du tableau DS. Ces chaînes combinent des espaces, des points circulaires (caractère shift Q) et des caractères de déplacement de curseur permettant d'atteindre les neuf positions possibles dans le carré.

Ce dessin peut également être réalisé sur les autres systèmes. La version Apple (qui n'est pas montrée ici) utilise un tableau à deux dimensions D$ (6,5) à la place du tableau à une dimension D$ (6) de la version Pet. Dans la version Apple, il n'y a pas de chaîne BX$ pour l'encadrement. A la place, chaque image est constituée par cinq lignes, qui comprennent et le pourtour du cadre et les points. C'est ainsi que, sur Apple, le dessin du premier dé est réalisé par les instructions suivantes :

```
FOR LX = L1 TO L1 + 4
    LL = LX : CC = C1 : GOSUB cursor
    PRINT D$(D1, LX - L1 + 1);
    NEXT LX
```

Des instructions analogues servent à effacer l'image dans la version Apple de *effacedés*. Les côtés du cadre sont faits de tirets et de deux points sur Apple, alors que les points du dé sont des astérisques. La version TRS-80 (citée dans les notes au bas des figures) est semblable à la version Apple, mais une version TRS-80 proche de la version Pet, est également réalisable.

L'affichage et l'effacement de l'enjeu, du résultat, de la ligne du coup en cours et de la ligne des statistiques, sont faciles à comprendre. L'effacement se fait en utilisant des chaînes de blancs. Se reporter aux routines *effacenjeu* (fig. 8.23), *effaceresult* (fig. 8.24) et

---

* La touche « shift » est celle, utilisée sur les machines à écrire, pour obtenir les caractères majuscules ou les caractères gravés dans la partie supérieure des touches.

effacestatis (fig. 8.25). Il n'est pas nécessaire d'effacer la ligne du coup en cours. Cette ligne est écrite en surimpression et les nouveaux caractères recouvrent toujours la totalité des précédents (fig. 8.20).

Rapidité et continuité dans le déroulement du jeu

La rapidité et la continuité dans le déroulement de l'action sont des éléments de la plus haute importance dans la conception d'un jeu. La rapidité est réalisée dans le programme de Craps par :

— Des entrées mono-caractères, sans signaux d'appel intempestifs.

— La disparition automatique des dés et de certains messages après un délai réglable.

— L'existence de plusieurs zones d'affichage qui peuvent changer à chaque fois qu'une touche est frappée.

```
# Analyser les codes entrés
codentre  IF case
              X$ = " " THEN IC = 2       # barre d'espacement pour jouer
              X$ = " N " THEN IC = 3     # nouveau joueur
              X$ = " Q " THEN IC = 4     # un joueur sort
              X$ = " S " THEN IC = 5     # enjeux secondaires
              X$ = " P " THEN IC = 6     # passer les dés
          else
              IC = 1                     # par défaut : considéré comme code
                                         enjeu
          RETURN

160 REM CODENTRE
170 IF X$=" "THEN IC=2:RETURN
180 IF X$="N" THEN IC=3:RETURN
190 IF X$="Q" THEN IC=4:RETURN
200 IF X$="S" THEN IC=5:RETURN
210 IF X$="P" THEN IC=6:RETURN
220 IC=1:RETURN
```

La routine « codentré » analyse le premier caractère entré pour la partie. Si le caractère est identifié, la routine affecte alors à IC la valeur adéquate, valeur qui sera utilisée dans le branchement ON ... GOSUB de la routine principale. Dans le cas contraire, IC reçoit la valeur par défaut, valeur qui fera que le caractère entré sera traité comme une commande de fixation de l'enjeu par la routine « enjeu ».

Figure 8.9 : Décodage du Premier Caractère Entré

La commutation de contexte

Notre version de Craps fournit un exemple simple de commutation de contexte. Celle-ci est un des points les plus importants de programmes beaucoup plus complexes tels que les systèmes en temps partagé. En appliquant le vocabulaire de l'informatique à la table de craps, nous pourrions dire que chaque joueur représente un *traitement*. Votre traitement particulier est constitué par vos actions quand c'est à vous de jouer : parier et jeter les dés. Les traitements des autres joueurs sont constitués par leurs propres actions pendant la durée de leur tour.

Dans un casino, la table, les dés, la position de l'enjeu sur le tapis, constituent les *ressources partagées*. Un seul traitement peut à un instant donné utiliser ces ressources (ce qui revient à dire que les dés ne peuvent être lancés que par un seul joueur) ; les autres traitements sont *interrompus* pendant ce temps.

Dans notre programme de Craps, le clavier, l'écran, et les programmes sont les ressources partagées, et les traitements sont les diverses interactions des joueurs avec ces ressources. Les seuls éléments dans le programme qui changent d'un joueur à l'autre sont le nom du joueur et ses statistiques. Ces éléments constituent le *contexte* dans lequel les programmes sont exécutés quand le traitement correspondant à un joueur donné, est activé. Plus précisément le contexte est constitué par les variables WN (gains/pertes), BT (montant de l'enjeu), BR (meilleure suite de gains), OB (enjeu initial) et CP (numéro du joueur en cours — utilisé comme indice du tableau des noms et comme un des indices de celui des statistiques).

La commutation de contexte a lieu dans la routine *passer* de la figure 8.30. Dans le cas présent, la majeure partie de la machine et du programme est partagée, les seules ressources affectées en propre aux traitements étant les éléments qui leur sont attribués dans les tableaux des noms et des statistiques.

Cette commutation simple est à la base de tout *multitraitement*. Dans un programme plus complexe, comme un système en temps partagé, chaque tâche peut très bien avoir son propre écran et son propre clavier. Dans ce cas, la commutation interviendrait beaucoup plus souvent, car à chaque fois que le programme aurait à attendre une entrée au clavier pour un traitement donné, l'exécution serait transférée au traitement suivant. Vous ne vous rendriez pas compte que le programme est en train de traiter les enjeux et le coup de dés de quelqu'un d'autre. En fait, vous auriez l'illusion d'avoir la machine pour votre usage exclusif. Quand vous attendez la réponse du programme, il se consacre à vous totalement, mais quand vous pensez

qu'il vous attend, en fait il ne vous accorde pas la moindre attention. (Ceci ressemble à ce que faisait le grand chanteur Chaliapin qui prétendait qu'en respirant au moment où l'on ne s'y attendait pas et qu'en ne respirant pas lorsqu'on s'y attendait, il était capable de donner l'impression qu'il ne respirait jamais.)

Il resterait bien des choses à dire sur le temps partagé et les autres systèmes de multitraitement, mais la simple commutation de contexte illustré par le programme de Craps, vous donne une idée de la façon dont fonctionnent de tels systèmes.

```
# Jouer une partie

partie  GOSUB lancement                          # lancement initial
        IF D = 2, 3 or 12 THEN                   # 2, 3 ou 12 font perdre
             { GOSUB perdu : RETURN }
        else IF D = 7 or 11 THEN                 # 7 ou 11 font gagner
             { GOSUB gagne : RETURN }
        else {
             PT = D : TH = 1                     # préciser valeur point, nom-
                                                 bre de lancements
             RS$ = " LE POINT EST " + STR$(PT)   # annoncer le « résultat » du
                                                 lancement
             GOSUB resultat
             }
        GOSUB statispartie                       # afficher le point et les lan-
                                                 cements
        repeat {                                 # essayer de faire le point
             GOSUB uncar                         # attendre que le joueur
                                                 appuie sur une touche
             GOSUB lancement : TH = TH + 1       # generation d'un lancement
             GOSUB statispartie                  # afficher le point et les lan-
                                                 cements
             IF D = PT THEN
                  { GOSUB gagne : break }        # faire le point fait gagner
             else IF D = 7 THEN
                  { GOSUB perdu : break }        # 7 fait perdre
             else {
                  RS$ = STR$(D) + " — LANCER A NOUVEAU "
                  GOSUB resultat
                  }
             }
        RETURN
```

La routine « partie » traite une série complète de lancements. Un 7 ou un 11 initial fait gagner immédiatement ; 2, 3 ou 12 font perdre tout de suite. Tout autre résultat devient le « point » et les dés sont lancés à nouveau jusqu'à ce que le point ou 7 soit tiré.

Figure 8.10 : Une Suite de Lancements de Dés

```
230  REM PARTIE
240  GOSUB 370
250  IF D=2 OR D=3 OR D=12 THEN GOSUB 490:RETURN
260  IF D=7 OR D=11 THEN GOSUB 420:RETURN
270  PT=D:TH=1
280  RS$="LE POINT EST"+STR$(PT):GOSUB 1020
290  GOSUB 980
300* GOSUB 2340:GOSUB 370
310  TH=TH+1:GOSUB 980
320  IF D=PT THEN GOSUB 420:GOTO 360
330  IF D=7 THEN GOSUB 490:GOTO 360
340  RS$=STR$(D)+"—LANCER A NOUVEAU":GOSUB 1020
350  GOTO 300
360  RETURN
```

* Pour les versions Apple et TRS-80, 2340 est remplacé par 2660 (voir figure 8.8).

Figure 8.10a : La Partie *(BASIC Réel)*

```
# Lancer les dés
lancement  D1 = INT(6 * RND(1) + 1)    # valeur du premier dé
           D2 = INT(6 * RND(1) + 1)    # valeur du second dé
           GOSUB dessinedes            # afficher l'image des dés
           D = D1 + D2                 # valeur totale du lancement
           RETURN
```

```
370   REM LANCEMENT
380*  D1=INT(6*RND(1)+1)
390*  D2=INT(6*RND(1)+1)
400** GOSUB 2240
410   D=D1+D2:RETURN
```

La routine « lancement » tire au hasard deux nombres compris entre un et six (les valeurs portées par les faces des deux dés). Le sous-programme « dessinedés », qui affiche les dés, se trouve en fin de programme avec « effacedés », « attente » et « initialisation ». Ces quatre routines sont réalisées de façon différente suivant l'ordinateur individuel concerné. Le reste du programme est pratiquement identique quelle que soit la version (Pet, Apple, ou TRS-80).

* Sur TRS-80, RND(1) est remplacé par RND(0).
** Pour les versions Apple et TRS-80, 2240 est remplacé par 2500 (voir figure 8.8).

Figure 8.11 : Lancement des Dés

Tirage au hasard

Il importe que la valeur des dés soit imprévisible. Malheureusement, étant donné qu'un ordinateur et un programme donnés se comportent de façon identique à chaque mise sous tension, la suite des nombres générés par la fonction RND peut être prévue. Le procédé qui permet au programmeur de cacher au joueur cette prévisibilité, est appelé *tirage aléatoire* (« randomization »). La technique, utilisée dans la routine *uncar* (fig. 8.37) pour assurer un tirage aléatoire, consiste à appeler la fonction RND (nombres au hasard) un nombre imprévisible de fois. Le nombre d'appels est imprévisible parce que ces appels font partie d'une boucle qui attend que le joueur appuie sur une touche. Plus longue est l'attente, plus grand est le nombre d'appels à la fonction RND. La durée d'une itération est si brève (quelques millièmes de secondes au maximum) que le joueur ne peut pas, en pratique, maîtriser le nombre d'itérations.

```
# Traiter un gain

gagne  WN = WN + BT                                # ajouter aux gains
       RD = 2 * BT                                 # montant à laisser courir
       PS = PS + 1                                 # un gain consécutif de plus
       BR = max(BR, PS)                            # meilleure suite réalisée
       ZP = CP : GOSUB sauverstatis                # mettre à jour tableaux sta-
                                                   tistiques
       RS$ = "C'EST GAGNE !" : GOSUB resultat      # publier résultat
       GOSUB attente                               # donner au joueur le temps
                                                   de le lire
       RETURN

420 REM GAGNE
430 WN=WN+BT
440 RD=2*BT:PS=PS+1
450 IF PS>BR THEN BR=PS
460 ZP=CP:GOSUB 1890
470 RS$="C'EST GAGNE!":GOSUB 1020
480 GOSUB 1380:RETURN
```

Quand le joueur gagne, la routine « gagné » est appelée pour mettre à jour les statistiques et annoncer le résultat.

Figure 8.12 : Actions à Entreprendre après un Gain

Cette façon de procéder marche très bien sur Pet et TRS-80, mais pas sur Apple, car la saisie mono-caractère sur Apple attend qu'un caractère soit effectivement entré pour continuer. En conséquence, on n'utilise la boucle qu'une seule fois. Pour faire des tirages au hasard que le joueur ne puisse pas contrôler, vous devez utiliser le « manche à balai » de l'Apple. La séquence suivante donnera le résultat désiré :

1. Dire au joueur d'appuyer sur le bouton du manche à balai.
2. Attendre qu'on cesse d'appuyer sur le bouton.
3. Appeler RND de façon continue et tester la position du bouton jusqu'à ce qu'on appuie dessus.

L'opération 2 empêche le joueur de contrôler le nombre d'appels à RND, en appuyant sur le bouton avant que le programme ne commence à vérifier la position de ce dernier.

```
#  Traiter une perte

perdu  WN = WN - BT                                  # soustraire des gains
       RD = OB                                       # montant à laisser cou-
                                                     rir = ancien enjeu
       PS = 0                                        # mise à zéro de la série
       ZP = CP : GOSUB sauverstatis                  # mettre à jour tableau
                                                     statistiques
       RS$ = " DESOLE, C'EST PERDU " : GOSUB resultat  # publier résultat
       GOSUB attente                                 # donner au joueur le
                                                     temps de le lire
       GOSUB passer                                  # passer les dés au joueur
                                                     suivant
       RETURN

490  REM PERDU
500  WN=WN-BT
510  RD=OB:PS=0
520  ZP=CP:GOSUB 1890
530  RS$="DESOLE, C'EST PERDU":GOSUB 1020
540  GOSUB 1380:GOSUB 1560
550  RETURN
```

Quand le joueur perd, la routine « perdu » est appelée pour mettre à jour les statistiques, annoncer le résultat et passer les dés au joueur suivant.

Figure 8.13 : Ce qu'il Faut Effectuer après une Perte

```
# Demander l'enjeu

enjeu   repeat {

            GOSUB codenjeu                # analyser caractère entré

            ON B GOSUB
                double,                   # doubler l'enjeu
                laissercourir             # laisser courir les gains
                gainperte                 # enjeu est le montant des gains ou pertes
                ancien                    # retour à l'ancien enjeu
                nombre                    # entrer un enjeu numérique

            IF BT < = 0 THEN {            # code enjeu non valide
                LL = BL
                CC = BC
                GOSUB curseur
                PRINT "DESOLE, ..."
                GOSUB uncar               # entrer un autre caractère
                }

            else IF BT > MX THEN {        # enjeu au-dessus du maxi
                LL = BL
                CC = BC
                GOSUB curseur
                PRINT " LA LIMITE DE L'ETABLISSEMENT EST "; MX
                                          # mentionner le maximum « maison »
                BT = MX                   # fixer l'enjeu au maximum
                IF OB > MX THEN
                    OB = MX
                GOSUB attente             # donner au joueur le temps de lire
                }

            } until (0 < BT < = MX)


        RETURN
```

La routine « enjeu » analyse le caractère qui lui a été transmis dans la variable X$ par la routine principale. Elle appelle l'une des routines « double », « laissercourir », « gainperte », « ancien » ou « nombre » et contrôle alors le montant de l'enjeu qui lui est renvoyé.

Figure 8.14 : Mettre la Commande d'Enjeu sur la Bonne Voie

```
560   REM ENJEU
570   GOSUB 700
580   ON B GOSUB 860,880,900,920,770
590   IF 0<BT AND BT<=MX THEN 690
600   IF BT<=MX THEN 650
610   LL=BL:CC=BC:GOSUB 1270
620*  PRINT "LA LIMITE DE L'ETABLISSEMENT EST";MX;"";
630   BT=MX:IF OB>MX THEN OB=MX
640   GOSUB 1380:GOTO 690
650   LL=BL:CC=BC:GOSUB 1270
660   PRINT "DESOLE, ENTREZ UN NOUVEAU CODE ENJEU S'IL VOUS PLAIT";
670** GOSUB 2340
680   GOTO 570
690   RETURN
```

* La version Apple de cette ligne n'en diffère que par le nombre d'espaces dans la constante littérale.
** Voir la note de la figure 8.8.

Figure 8.14a : ***L'Enjeu*** *(BASIC Réel)*

```
# Interpréter le code d'enjeu

codenjeu  IF case
               X$ = " D "             THEN B = 1    # doubler la mise
               X  = " R "             THEN B = 2    # laisser courir le dernier gain
               X$ = " W " or " L "    THEN B = 3    # enjeu = total gagné ou perdu
               X$ = " O "             THEN B = 4    # enjeu = enjeu initial
               X$ = " B "             THEN B = 5    # le joueur va entrer un nombre
          else                             B = 4    # par défaut, enjeu initial
          RETURN

700   REM CODENJEU
710   IF X$="D" THEN B=1:RETURN
720   IF X$="R" THEN B=2:RETURN
730   IF X$="W" OR X$="L" THEN B=3:RETURN
740   IF X$="O" THEN B=4:RETURN
750   IF X$="B" THEN B=5:RETURN
760   B=4:RETURN
```

La routine « codenjeu » s'empare des codes que la routine « codentré » a laisser passer et les classe. Si « codenjeu » n'arrive pas non plus à identifier la commande, cette dernière est traitée comme la commande « ancien enjeu ». Ce traitement par défaut est peu dangereux. Le contrôle retourne au programme principal, de telle sorte que le joueur qui est entré dans le sous-programme « enjeu » par inadvertance, n'est pas obligé de frapper au clavier un code d'enjeu valide.

Figure 8.15 : ***Analyse de la Commande d'Enjeu***

```
# Saisir un nombre

nombre      AA = 0                              # initialiser le nombre
            GOSUB signalenjeu                   # demande enjeu
            repeat {
                 GOSUB uncar                    # caractère suivant
                 NN = ASC(X$) – ASC("0")        # en faire un chiffre
                 IF NN < 0 OR NN > 9 THEN       # si pas un chiffre, alors terminé
                      break
                 PRINT X$;                      # répercuter le chiffre
                 AA = 10 * AA + NN              # faire cumul du nombre
                 }
            BT = AA                             # fixer montant enjeu
            OB = BT                             # fixer montant « initial »
            RETURN


# Signal demander entrée

signalenjeu GOSUB effacenjeu
            PRINT "ENJEU : "
            RETURN
```

```
770  REM NOMBRE
780  AA=0:GOSUB 940
790* GOSUB 2340
800  NN=ASC(X$)–ASC("0")
810  IF NN<0 OR NN>9 THEN 840
820  PRINT X$;:AA=10*AA+NN
830  GOTO 790
840  BT=AA:OB=BT
850  RETURN

940  REM SIGNALENJEU
950  GOSUB 1180:PRINT "ENJEU:";:RETURN
```

La routine « nombre » et son sous-programme « signalenjeu » sont un exemple de programmation médiocre. (Pouvez-vous expliquer pourquoi ?) Une autre solution est présentée en figure 8.17.

---

* Dans les versions Apple et TRS-80, 2340 est remplacé par 2660 (voir figure 8.8).

Figure 8.16 : Saisie d'un Enjeu Numérique

```
# Autre version pour saisie enjeu numérique

nombre  repeat {
            GOSUB effacenjeu                          # effacer ligne d'enjeu
            PRINT "ENJEU :";: GOSUB entreechaîne      # entrée des caractères
                                                      jusqu'à frappe un RETURN
            IF XX$ =" " THEN                          # porte de sortie
                RETURN
            else
                NN = VAL(XX$)                         # convertir en nombre
            } until (NN <> 0)                         # n'accepter ni 0 ni non-
                                                      numérique
        BT = NN : OB = BT                             # fixer montant enjeu et
                                                      enjeu « initial »
        RETURN
```

Cette routine a le mérite de ne pas réinventer l'eau chaude. En effet, elle fait appel d'une part à la routine « entréechaîne » du programme du Jumelage, d'autre part à la fonction VAL incorporée au BASIC.

Figure 8.17 : Saisie d'un Enjeu Numérique *(une meilleure solution)*

```
# Routine fixant les enjeux

double          BT = 2 * BT : RETURN       # doubler la mise
laissercourir   BT = RD : RETURN           # laisser courir les gains
gainperte       BT = ABS(WN) : RETURN      # l'enjeu est le montant des gains ou
                                           pertes
ancien          BT = OB : RETURN           # retour à l'enjeu initial

    860  REM DOUBLE
    870  BT=2*BT:RETURN
    880  REM LAISSERCOURIR
    890  BT=RD:RETURN
    900  REM GAINPERTE
    910  BT=ABS(WN):RETURN
    920  REM ANCIEN
    930  BT=OB:RETURN
```

Voilà les petites routines appelées par « enjeu » poursuivant l'option choisie pour le montant du pari.

Figure 8.18 : Fixation de l'Enjeu

```
# Début de partie

depart   GOSUB effacecran      # effacer l'écran
         GOSUB statis          # afficher la ligne des statistiques et le nom du joueur
         RETURN

   960  REM DEPART
   970  GOSUB 1250:GOSUB 1050:RETURN
```

La routine « départ » efface l'écran et affiche les informations de base concernant le joueur : nom, gains, montant de l'enjeu, etc... Aucune information n'est fournie sur telle ou telle partie.

Figure 8.19 : Le Début de la Partie

```
# Afficher ligne de la partie

statispartie   LL = GL : CC = GC : GOSUB curseur
               PRINT " POINT : "; PT; " LANCEMENTS : "; TH;
               RETURN

    980   REM STATISPARTIE
    990   LL=GL:CC=GC:GOSUB 1270
   1000*  PRINT "POINT:";PT;"LANCEMENTS:";TH;
   1010   RETURN
```

La routine « statispartie » affiche la ligne qui contient le « point » du joueur ainsi que le nombre de lancements déjà effectués. « Statispartie » est appelée par « partie ». En premier lieu « Statispartie » positionne le curseur, puis affiche les informations. Etant donné que la valeur de PT (le point) reste constante pendant la partie, la seule partie qui change est la valeur de TH, nombre de lancements.

* La version Apple de cette ligne n'en diffère que par le nombre d'espaces dans la constante littérale.

Figure 8.20 : Affichage du Point et des Lancements

Afficher le résultat du lancement — RS$

```
resultat  GOSUB effaceresult     # « blanchir » la ligne de résultat
          PRINT RS$              # afficher le résultat
          GOSUB attente          # donner au joueur temps de voir
          GOSUB effacedes        # ratisser les dés
          RETURN
```

```
1020   REM RESULTAT
1030   GOSUB 1150
1040*  PRINT RS$;:GOSUB 1380:GOSUB 2300:RETURN
```

La routine « résultat » affiche le résultat du lancement des dés. La chaîne RS$ est transmise à cette routine par le programme appelant (« partie », « gagné », « perdu »). RS$ est porteur de messages du genre « C'EST GAGNÉ » ou « LE POINT EST 8 » ou « 6-LANCER A NOUVEAU ». Le fait de « blanchir » la ligne de résultat, non seulement donne la certitude que le résultat précédent est complètement effacé, mais aussi produit un clignotement perceptible de la ligne, même dans le cas où le nouveau résultat et l'ancien sont identiques. Ce clignotement permet au joueur de savoir que le programme vient de terminer le traitement du coup.

Après l'affichage de la ligne de résultat, le programme attend un court instant (la durée de cette attente est précisée dans la routine d'initialisation où est fixée la valeur de la variable DC) ; puis l'image des dés est effacée de l'écran.

Notons que ce court sous-programme entraîne trois modifications distinctes et remarquables dans la présentation de l'écran :

— la ligne est « blanchie »
— le nouveau résultat apparaît
— les dés sont effaçés.

Cette séquence de changements retient l'attention du joueur et rend le jeu plus absorbant.

---

* Dans les versions Apple et TRS-80, 2300 est remplacé par 2580 (voir figure 8.8).

Figure 8.21 : Affichage du Résultat du Lancement des Dés

```
# Afficher statistiques du joueur

statis  GOSUB effacestatis                      # « blanchir » la zone statistiques
        IF WN < 0 THEN                          # annoncer GAGNE ou PERDU
             PRINT " PERDU ",
        else
             PRINT " GAGNE ",
        PRINT " ENJEU ", " CETTE SERIE ", " MEILLEURE SERIE "
                                                # afficher reste du titre
        PRINT ABS(WN), BT, PS, BR ;"   ";       # afficher les valeurs
        IF CP > 0 THEN {                        # s'il y a un joueur, en afficher le nom
             CC = NC : LL = NL : GOSUB curseur
             PRINT NM$(CP) ;
             }
        RETURN
```

```
1050   REM STATIS
1060   GOSUB 1210
1070*  IF WN<0 THEN PRINT "PERDU",:GOTO 1090
1080*  PRINT "GAGNE",
1090*  PRINT "ENJEU","CETTE SERIE","MEILLEURE SERIE"
1100*  PRINT ABS(WN),BT,PS,BR;"   ";
1110   IF CP=0 THEN 1140
1120   CC=NC:LL=NL:GOSUB 1270
1130   PRINT NM$(CP);
1140   RETURN
```

La routine « statis » affiche les statistiques concernant le joueur — montant gagné ou perdu, enjeu, nombre de gains consécutifs (cette série) et le plus grand nombre de gains consécutifs jamais réalisé (meilleure série). Les valeurs affichées sont celles des variables simples WN, BT, PS et BR ; ces variables sont valorisées à partir des éléments propres au joueur figurant dans le tableau des statistiques. Ce transfert des valeurs est effectué par la routine « passer » quand le tour du joueur arrive.

Il est plus élégant d'afficher GAGNE ou PERDU selon que WN est positif ou négatif, que de se contenter d'afficher quelque chose comme GAINS : − 100 quand le joueur est en train de perdre.

* Dans la version Apple, l'intervalle entre les quatre informations et leurs légendes est obtenu par TAB(10), TAB(20) et TAB(30) et non par des virgules. La raison de cette différence est que l'utilisation de virgules dans les instructions PRINT du BASIC Apple, ne permet l'affichage que de 3 données par ligne au lieu de quatre.

Figure 8.22 : Affichage des Statistiques du Joueur

Nettoyage avant entrée de l'enjeu

```
effacenjeu   GOSUB depart                          # afficher écran au départ
             CC = BC : LL = BL : GOSUB curseur     # positionner le curseur
             RETURN
```

```
1180 REM EFFACENJEU
1190 GOSUB 960
1200 CC=BC:LL=BL:GOSUB 1270:RETURN
```

La routine « effacenjeu » prépare l'entrée de l'enjeu. Etant donné que rien d'important (à part les statistiques concernant le joueur) ne se trouve sur l'écran, « effacenjeu » utilise la solution facile qui consiste à effacer l'écran à réafficher les statistiques (tout cela est fait dans la routine « départ »), puis à repositionner le curseur à l'endroit convenable.

Figure 8.23 : Préparation avant Saisie de l'Enjeu

Effacer la ligne du résultat

```
effaceresult   CC = RC : LL = RL : GOSUB curseur   # positionner
               PRINT RC$;                          # « blanchir » et repositionner
               RETURN
```

```
1150 REM EFFACERESULT
1160 CC=RC:LL=RL:GOSUB 1270
1170 PRINT RC$;:RETURN
```

La routine « effaceresult », blanchit la ligne du résultat et remet le curseur en position pour l'affichage du nouveau résultat. La remise en position du curseur est obtenue par l'inclusion de caractères de déplacement du curseur dans la chaîne RC$. Ceci fait que cette routine est spécifique du Pet; mais la version pour Apple et TRS-80 n'est que peu différente. Sur ces derniers systèmes, RC$ ne contient pas de caractères de déplacement du curseur et après le PRINT RC$;, on répète la première ligne. La différence étant mineure, cette routine n'est pas incluse, en fin de programme, dans les routines spécifiques d'une configuration donnée. En réalité la solution Apple et TRS-80 marcherait très bien sur Pet, sans perte de performance. Les instructions BASIC pour l'Apple et le TRS-80 sont :

```
1150 REM EFFACERESULT
1160 CC=RC:LL=RL:GOSUB 1270:PRINT RC$
1170 CC=RC:LL=RL:GOSUB 1270:RETURN
```

Figure 8.24 : Effacement de la ligne de Résultat

Effacer la zone statistique sur l'écran

effacestatis CC = 0 : LL = SL : GOSUB curseur # première colonne de la première ligne statistique
PRINT BL$: PRINT BL$; # effacer deux lignes
CC = 0 : LL = SL : GOSUB curseur # reposition
RETURN

```
1210  REM EFFACESTATIS
1220  CC=0:LL=SL:GOSUB 1270
1230  PRINT BL$:PRINT BL$;
1240  CC=0:LL=SL:GOSUB 1270:RETURN
```

La routine « effacestatis » efface deux lignes (lignes SL et SL + 1). Le curseur est alors repositionné au début de la zone effacée, pour préparer l'affichage des statistiques par la routine « statis ».

Figure 8.25 : Effacement de la Zone des Statistiques

Effacer l'écran

effacecran PRINT " clr ";
RETURN

```
1250  REM EFFACECRAN
1260  PRINT CHR$(147);:RETURN
```

La routine « effacecran » efface l'écran. Ceci est la version Pet. Les versions Apple et TRS-80 sont légèrement différentes (la première ligne est remplacée par HOME sur Apple, par CLS sur TRS-80). Dans les trois cas, l'écran est effacé, et le curseur est envoyé dans le coin supérieur gauche. (Voir figure 2.12.)

Figure 8.26 : Effacement de l'Ecran

```
#  Mettre le curseur en ligne LL, colonne CC — Version Pet
curseur  PRINT " coin supérieur gauche ";
         CC = CC mod colonnes : LL = LL mod lignes
         IF CC <> 0 THEN                          # curseur à droite CC fois
              FOR ZZ = 1 TO CC
                   PRINT " à droite ";
                   NEXT ZZ
         IF LL <> 0 THEN                          # curseur vers le bas LL fois
              FOR ZZ = 1 TO LL
                   PRINT " vers le bas ";
                   NEXT ZZ
         RETURN

1270  REM CURSEUR
1280  PRINT CHR$(19);
1290  IF CC<0 THEN CC=CC+40:GOTO 1290
1300  IF CC>39 THEN CC=CC-40:GOTO 1300
1310  IF LL<0 THEN LL=LL+24:GOTO 1310
1320  IF LL>23 THEN LL=LL-24:GOTO 1320
1330  IF CC=0 THEN 1350
1340  FOR ZZ=1 TO CC:PRINT CHR$(29);:NEXT ZZ
1350  IF LL=0 THEN RETURN
1360  FOR ZZ=1 TO LL:PRINT CHR$(17);:NEXT ZZ:RETURN
```

Les versions Apple et TRS-80 ont les différences signalées à la figure 4.16.

Figure 8.27 : Positionnement du Curseur

```
#  Attendre suffisamment pour que le joueur ait le temps de lire
attente  FOR ZZ = 1 TO DC
              NEXT ZZ
         RETURN

1380  REM ATTENTE
1390  FOR ZZ=1 TO DC:NEXT   ZZ:RETURN
```

La routine « attente » fait tourner une boucle qui ne fait rien d'autre, un nombre de fois précisé par DC. La valeur de DC est fixée dans un ordre DATA de la routine « initialisation ». La durée réelle de l'attente est ce qui rend l'action du jeu plus ou moins rapide. La meilleure façon de déterminer la valeur de DC est de faire des essais successifs. Une autre méthode, plus précise consisterait à utiliser la variable TI particulière au Pet. (Voir figure 9.36.)

Figure 8.28 : Ne Rien Faire Pendant Quelque Temps

```
# Faire entrer un nouveau joueur
nouveau  IF NP > = MP THEN {                                # si pas de place,
             CC = BC : LL = BL : GOSUB curseur              # alors le dire
             PRINT "PLUS DE PLACE... ";
             GOSUB attente : RETURN
             }
         NP = NP + 1 : GOSUB effacecran                     # un joueur de plus
         INPUT "...VOTRE NOM, S'IL VOUS PLAIT"; NM$(NP)     # obtenir son nom
         PRINT : PRINT " MERCI. ";
         IF NP > 1 THEN {                                   # si ce n'est pas le
                                                            premier joueur
             PRINT "... APRES "; NMS(NP - 1)
             ZP = CP : GOSUB sauverstatis                   # sauver statisti-
                                                            ques en cours
             GOSUB misazero:ZP = NP:GOSUB sauverstatis      # donner au jou-
                                                            eur des statistiques
                                                            neuves
             ZP = CP : GOSUB chargerstatis                  # restaurer les sta-
                                                            tistiques en cours
             }
         else                                               # si le joueur est le
                                                            premier, les statis-
                                                            tiques
             { ZP = 1 : GOSUB sauverstatis : CP = 1 }       # en cours devien-
                                                            nent celles du jou-
                                                            eur nº 1
         GOSUB attente : GOSUB attente : RETURN             # laisser au joueur
                                                            le temps de lire

1400  IF NP<MP THEN 1450
1420  CC=BC:LL=BL:GOSUB 1270
1430  PRINT "PLUS DE PLACE A LA TABLE";
1440  GOSUB 1380:RETURN
1450  NP=NP+1:GOSUB 1250
1470* INPUT "POURRAIS JE AVOIR VOTRE NOM S'IL VOUS PLAIT";NM$(NP)
1480  PRINT:PRINT "MERCI.";
1490  IF NP<=1 THEN 1540
1500  PRINT"VOUS JOUEREZ APRES";NM$(NP-1)
1510  ZP=CP:GOSUB 1890
1520  GOSUB 1860:ZP=NP:GOSUB 1890
1530  ZP=CP:GOSUB 1920
1540  CP=1:ZP=1:GOSUB 1890
1550  GOSUB 1380:GOSUB 1380:RETURN
```

La routine « nouveau » fait rentrer un nouveau joueur dans le jeu. Le joueur se voit attribuer des statistiques initialisées et un tour pour jouer.

---

* La version Apple n'en diffère que par la valeur de la constante littérale.

Figure 8.29 : Entrée d'un Nouveau Joueur

```
# Passer les dés

passer  IF NP = 0 THEN RETURN             # si aucun joueur, c'est pas grave
        ZP = CP : GOSUB sauverstatis      # sauver statistiques du joueur en cours
        CP = (CP mod NP) + 1              # passer au joueur suivant
        ZP = CP : GOSUB chargerstatis     # charger les statistiques du joueur dont
                                          le tour commence
        PS = 0 : RETURN                   # on commence avec zéro partie gagnée

   1560  IF NP=0 THEN RETURN
   1580  ZP=CP:GOSUB 1890
   1590  CP=CP+1:IF CP>NP THEN CP=1
   1600  ZP=CP:GOSUB 1920:PS=0
   1610  RETURN
```

La routine « passer » fait passer les dés d'un joueur au suivant. Elle est appelée automatiquement à partir de « perdu » ou bien par une commande explicite entrée par le joueur.

Figure 8.30 : Les Dés Passent

```
# Afficher noms et gains/pertes de tous les joueurs

afficherjoueurs  GOSUB effacecran                           # effacer l'écran
                 PRINT " NUMERO ", " GAGNE/PERDU ", " NOM "
                                                            # affichage du titre
                 PRINT
                 FOR ZZ = 1 TO NP
                      PRINT ZZ, ST(1, ZZ), NM$(ZZ)          # afficher numéro
                      NEXT ZZ                               # gains/pertes, nom
                      PRINT : RETURN

   1620   GOSUB 1250
   1640*  PRINT:PRINT "NUMERO","GAGNE/PERDU","NOM":PRINT
   1650   FOR ZZ=1 TO NP
   1660*  PRINT ZZ,ST(1,ZZ),NM$(ZZ):NEXT ZZ
   1670   PRINT
   1680   RETURN
```

Le principal objet de la routine « afficherjoueurs » est de dévoiler les numéros affectés aux joueurs. Les joueurs peuvent alors se désigner par leurs numéros dans des commandes telles la commande Q (quit, quitter abandonner).

* La version Apple de ces lignes utilisent TAB(10) et TAB(20) au lieu de virgule pour régler l'espacement entre les éléments.

Figure 8.31 : Voici les Numéros des Joueurs

```
# Un joueur s'en va

abandon  GOSUB afficherjoueurs                        # montrer les noms et les
                                                      numéros

         repeat {
              INPUT " ...NUMERO, S'IL VOUS PLAIT ";   # demander le numéro
                                               QP
              IF QP = 0 THEN                          # 0 = porte de sortie
                   RETURN
              else IF 0 < QP < = NP THEN              # si numéro existant,
                   break                              # alors le supprimer
              else                                    # dans le cas contraire,
                   PRINT " S'IL VOUS PLAIT            # protester
                             REGARDEZ LA LISTE "
              }
         IF QP = CP THEN                              # s'il s'agit du joueur en
                                                      cours,
              GOSUB passer                            # passer les dés d'abord
         IF QP < NP THEN                              # si ce n'est pas le dernier
                                                      qu'on supprime
              FOR ZZ = QP TO NP - 1                   # décaler les suivants
                                                      d'un cran vers le haut
                   FOR YY = 1 TO nstatis
                        ST(YY, ZZ) = ST(YY, ZZ + 1)
                        NEXT YY
                   NM$(ZZ) = NM$(ZZ + 1)
                   NEXT ZZ
         NP = NP - 1                                  # un joueur en moins
         IF CP > = QP THEN                            # renuméroter joueur en
                                                      cours
              CP = CP - 1
         IF NP = 0 THEN                               # si aucun joueur,
              GOSUB misazéro                          # effacement statistiques
         RETURN
```

Cette routine permet de faire sortir un joueur du tour. Tous les principes habituels de la suppression s'y appliquent. (Voir figure 7.38.)

Figure 8.32 : ***Abandon*** *(Quitter la partie)*

```
1690  REM ABANDON
1700  GOSUB 1620
1710* INPUT "QUEL EST VOTRE NUMERO, S'IL VOUS PLAIT";QP
1720  IF QP=0 THEN RETURN
1730  IF QP<=0 OR QP>NP THEN PRINT "S'IL VOUS PLAIT REGARDEZ LA LISTE":
      GOTO 1710
1740  IF QP=CP THEN GOSUB 1560
1750  IF QP=NP THEN 1790
1760  FOR ZZ=QP TO NP-1:FOR YY=1 TO 4
1770  ST(YY,ZZ)=ST(YY,ZZ+1):NEXT YY
1780  NM$(ZZ)=NM$(ZZ+1):NEXT ZZ
1790  NP=NP-1:IF CP>=QP THEN CP=CP-1
1800  IF NP=0 THEN GOSUB 1860
1810  RETURN
```

* La version Apple de cette ligne n'en diffère que par la présence d'un point d'interrogation dans la constante littérale.

Figure 8.32a : Quitter la Partie *(BASIC Réel)*

```
# Paris secondaires (souche)

parisecondaire  CC = BC : LL = BL : GOSUB curseur   # curseur à la ligne d'enjeu
                PRINT " PARIS SECONDAIRES... "      # dire que ce n'est pas au
                                                    point
                GOSUB attente                       # laisser le joueur le lire
                RETURN

1820  REM PARISECONDAIRE
1830  CC=BC:LL=BL:GOSUB 1270
1840  PRINT "PARIS SECONDAIRES PAS ENCORE PROGRAMMES";
1850  GOSUB 1380:RETURN
```

Cette « souche » porte à la connaissance du joueur que la commande « S » n'a réellement aucun effet.

Figure 8.33 : Paris Secondaires

Manipuler le tableau des statistiques

```
misazero        WN = 0                # initialiser gains/pertes,
                BT = 0                # enjeu,
                BR = 0                # meilleure série
                OB = 0                # et enjeu initial
                RETURN

sauverstatis    ST(1, ZP) = WN        # stocker les statistiques du joueur numéro ZP
                ST(2, ZP) = BT
                ST(3, ZP) = BR
                ST(4, ZP) = OB
                RETURN

chargerstatis   WN = ST(1, ZP)        # aller chercher statis du joueur ZP
                BT = ST(2, ZP)
                BR = ST(3, ZP)
                OB = ST(4, ZP)
                RETURN
```

```
1860 REM MISAZERO
1870 WN=0:BT=0:BR=0:OB=0
1880 RETURN

1890 REM SAUVERSTATIS
1900 ST(1,ZP)=WN:ST(2,ZP)=BT:ST(3,ZP)=BR:ST(4,ZP)=OB
1910 RETURN

1920 REM CHARGERSTATIS
1930 WN=ST(1,ZP):BT=ST(2,ZP):BR=ST(3,ZP):OB=ST(4,ZP)
1940 RETURN
```

Cet ensemble de routines assure les fonctions fondamentales de gestion du fichier des statistiques. La routine « misazéro » initialise les variables de « travail » ; « sauverstatis » enregistre les valeurs de « travail » dans le tableau ; et « chargerstatis » charge les valeurs de « travail » à partir du tableau.

Figure 8.34 : Gestion du Fichier des Statistiques

Initialiser les constantes et les formats. Version pour le Pet

```
initialisation  READ SL : DATA lignestatis            # positions sur l'écran
                READ GL, GC : DATA lignepartie, colpartie
                READ RL, RC : DATA ligneresult, colresult
                READ L1, C1 : DATA ligne dé 1, col dé 1
                READ L2, C2 : DATA ligne dé 2, col dé 2
                READ BL, BC : DATA lignenjeu, colenjeu
                READ NL, NC : DATA lignenom, lignecolnom
                READ DC : DATA constante d'attente    # limite de la boucle dans
                                                      « attente »
                READ MX : DATA plafondmaison          # enjeu maximum admis
                FOR ZZ = 1 TO 6
                    READ D$(ZZ)
                    NEXT ZZ
                    DATA " face du dé pour 1 "        # configurations de points
                                                      3 × 3
                    DATA " face du dé pour 2 "        # (comprenant les
                    DATA " face du dé pour 3 "        # caractères de déplacement
                    DATA " face du dé pour 4 "        # du curseur)
                    DATA " face du dé pour 5 "
                    DATA " face du dé pour 6 "
                READ BX$ : DATA " cadres entourant    # encadrement de la face du
                                    les points "      dé
                READ BF$ : DATA " cadres et points    # gomme pour dés
                                  en « blanc »"
                READ RC$ : DATA " ligne de résultat   # gomme pour ligne de résul-
                                     en blanc "       tat
                READ BL$ : DATA " ligne de blancs "   # gomme pour ligne complète
                WN = 0 : PS = 0 : BT = 0 : BR = 0     # initialisation des variables
                NP = 0 : CP = 0
                MP = maxjoueurs
                DIM NM$ (maxjoueurs)                  # noms des joueurs
                    ST (nstatis, maxjoueurs)          # statistiques des joueurs
                RETURN
```

La routine « initialisation » initialise les constantes, variables et tableaux utilisés dans tout le programme. Les chaînes de caractères qui forment les dés dans le tableau D$ et les chaînes BX$ et BF$ qui servent à effacer les dés et leurs cadres, sont toutes basées sur les caractères de déplacement de curseur reconnus par le Pet. Les routines utilisant ces chaînes sont données en fin de programme après la présente routine. Les versions pour les autres systèmes sont complètement différentes.

Figure 8.35 : Initialisation

```
1950  REM INITIALISATION
1960  READ SL:DATA 20
1970  READ GL,GC:DATA 16,9
1980  READ RL,RC:DATA 11,10
1990  READ L1,C1:DATA 1,10
2000  READ L2, C2:DATA 3,17
2010  READ BL,BC:DATA 0,3
2020  READ NL,NC:DATA 23,14
2030  READ DC:DATA 200
2040  READ MX:DATA 500
2050  FOR XX=1 TO 6:READ D$(XX):NEXT XX
2060* DATA "(voir note)"
2070* DATA "(voir note)"
2080* DATA "(voir note)"
2090* DATA "(voir note)"
2100* DATA "(voir note)"
2110* DATA " (see note)"
2120  READ BX$
2130* DATA " (voir note)"
2140* READ BF$:DATA "(voir note)"
2150  READ C1$:DATA "  "
2160* READ C2$:DATA "(voir note)"
2170  RC$=C1$+C2$
2180  READ BL$
2190  DATA "                                "
2200  GOSUB 1860:PS=0
2210  NP=0:MP=9:CP=0
2220  DIM NM$(9),ST(4,9)
2230  RETURN
```

Voici les instructions BASIC réelles de la version Pet de la routine d'initialisation de la figure 8.35. Les versions Apple et TRS-80 sont présentées à la figure 8.36a.

---

* Les lignes 2060 à 2110 contiennent le dessin des faces des dés. Chacun est composé de 3 rangées d'espaces et de points. La première et la deuxième rangée sont dans les deux cas suivis par un caractère « curseur bas » et 3 caractères « curseur arrière ». Chaque chaîne contient, de ce fait, 17 caractères.

La ligne 2130 contient « l'encadrement » du dé. La chaîne contient une ligne « supérieure » composée d'un espace de 3 soulignés et d'un espace, 3 « côtés » composés de la barre verticale droite, de 3 espaces et de la ligne verticale gauche, et d'une ligne du « bas » composée de 1 espace et de 3 surlignés. La ligne supérieure et chacun des côtés est suivi par 5 caractères « curseur arrière » et 1 caractère « curseur bas ». La ligne du bas est suivie par 3 caractères « curseur arrière » et 3 caractères « curseur haut » (ce qui ramène le curseur dans le coin supérieur gauche de l'encadrement). La chaîne contient, de ce fait 55 caractères.

La ligne 2140 contient la chaîne qui efface les dés et qui comporte 5 rangées de 5 espaces chacune. Les 4 premières lignes sont chacune suivies de 5 « curseur arrière » et de 1 « curseur bas ».

La chaîne de la ligne 2160 ne comporte que des « curseurs arrière ». La longueur de la chaîne est la même que celle de la chaîne de « blancs » de la ligne 2150 (18 caractères).

Figure 8.36 : ***Initialisation*** *(BASIC Réel)*

```
1950    REM INITIALISATION
1960*   READ SL:DATA 20
1970*   READ GL,GC:DATA 16,9
1980*   READ RL,RC:DATA 11,10
1990    READ L1,C1:DATA 1,10
2000*   READ L2,C2:DATA 3,17
2010    READ BL,BC:DATA 0,3
2020*   READ NL,NC:DATA 23,14
2030*   READ DC:DATA 400
2040    READ MX:DATA 500
2050    FOR ZZ=0 TO 6:FOR XX=1 TO 5
2060    READS D$(ZZ,XX)
2070    NEXT XX:NEXT ZZ
2080    DATA "   ","   ","   ","   ","   "
2130    DATA "---",":   :",": * :",":   :","---"
2180    DATA "---",":*  :",":   :",":  *:","---"
2230    DATA "---",":*  :",": * :",":  *:","---"
2280    DATA "---",":**:",":   :",":  *:","---"
2330    DATA "---",":**:",": * :",":**:","---"
2380    DATA "---",":**:",":**:",":**:","---"
2440    READ RC$:DATA "                    "
2450**  READ BL$:DATA "                                        "
2460    GOSUB 1860:PS=0
2470    NP=0:MP=9:CP=0
2480    DIM NM$(9),ST(4,9)
2490    RETURN
```

Voici le listing en BASIC réel correspondant aux versions Apple et TRS-80 de Craps. La raison de l'écart important entre les numéros de lignes allant de 2080 à 2440 est que dans une version antérieure, chacune des lignes 2080, 2130, 2180, 2230, 2280, 2330 et 2380 était remplacée par cinq lignes différentes.

---

* Dans la version TRS-80 de ces lignes SL vaut 13, GL vaut 11, GC vaut 21, RL vaut 10, RC vaut 22, C2 vaut 25, NL vaut 15, NC vaut 26 et DC vaut 100.

** Sur TRS-80 cette ligne comporte 63 blancs, l'écran ayant une largeur de 64 colonnes.

Figure 8.36a : ***Initialisation*** *(Versions Apple et TRS-80)*

Attendre l'entrée d'un caractère et en même temps faire tourner le générateur de nombres aléatoires

```
uncar   repeat {

           GET X$                 # y a-t-il un caractère à prendre ?
           ZZ = RND(1)            # chatouiller le générateur de nombres aléa-
                                  toires
           } until (X$ <> "")     # jusqu'à ce qu'un caractère soit frappé

        RETURN
```

```
2340   REM UNCAR
2350*  GET X$:IF X$<>""THEN RETURN
2360*  ZZ=RND(1):GOTO 2350
```

« Uncar » est la routine de saisie qui apparaît partout dans ce livre, mais avec en plus des appels réitérés à la fonction RND. Etant donné que le nombre d'itérations de la boucle est probablement grand et que le joueur ne peut pas le maîtriser, « uncar » assure un tirage véritablement au hasard.

En raison du mode de fonctionnement de l'instruction GET sur Apple, un tirage véritablement au hasard ne peut pas être obtenu par la version Apple de cette routine. Cette version Apple se trouve en figure 2.11.

Les versions TRS-80 et Apple de cette routine vont de la ligne 2660 à la ligne 2680.

---

* Pour ces lignes, dans la version TRS-80, GET X$ est remplacé par X$=INKEY$ et RND(1) est remplacé par RND(0).

Figure 8.37 : Entrée Mono-caractère

Dessiner les dés — Version pour le Pet

```
dessinedes  LL = L1              # positionner le curseur
            CC = C1
            GOSUB curseur
            PRINT BX$ ; D$(D1) ;  # dessiner le premier dé
            LL = L2              # positionner le curseur
            CC = C2
            GOSUB curseur
            PRINT BX$ ; D$(D2) ;  # dessiner le second dé
            RETURN
```

```
2240 REM DESSINEDES
2250 LL=L1:CC=C1:GOSUB 1270
2260 PRINT BX$;D$(D1);
2270 LL=L2:CC=C2:GOSUB 1270
2280 PRINT BX$;D$(D2);
2290 RETURN
```

En raison de l'inclusion de caractères de contrôle du curseur dans les chaînes de caractères BX$ et D$, la routine « dessinedés » est très simple. Les versions Apple et TRS-80 sont complètement différentes.

Les versions Apple et TRS-80 utilisent un tableau D$ à deux dimensions. Les instructions BASIC sont les suivantes :

```
2500 REM DESSINEDES
2510 FOR LX=L1 TO L1+4
2520 LL=LX:CC=C1:GOSUB 1270
2530 PRINT D$(D1,LX-L1+1);:NEXT LX
2540 FOR LX=L2 TO L2+4
2550 LL=LX:CC=C2:GOSUB 1270
2560 PRINT D$(D2,LX-L2+1);:NEXT LX
2570 RETURN
```

Figure 8.38 : L'Affichage des Dés

Ratissage des dés — Version pour le Pet

```
effacedes   CC = C1
            LL = L1
            GOSUB curseur
            PRINT BF$;
            CC = C2
            LL = L2
            GOSUB curseur
            PRINT BF$;
            RETURN
```

```
2300  REM EFFACEDES
2310  CC=C1:LL=L1:GOSUB 1270:PRINT BF$;
2320  CC=C2:LL=L2:GOSUB 1270:PRINT BF$;
2330  RETURN
```

La routine « effacedés » efface l'image des dés (dessinés par « dessinedés »). Elle est particulière au Pet pour la même raison que « dessinedés ». Les versions Apple et TRS-80 sont complètement différentes. En voici les instructions BASIC :

```
2580  REM EFFACEDES
2590  FOR LX=L1 TO L1+4
2600  LL=LX:CC=C1:GOSUB 1270
2610  PRINT D$(0,LX-L1+1);:NEXT LX
2620  FOR LX=L2 TO L2+4
2630  LL=LX:CC=C2:GOSUB 1270
2640  PRINT D$(0,LX-L2+1);:NEXT LX
2650  RETURN
```

La fonction assurée par « effacedés » est extrêmement importante. Les dés sont effacés de l'écran sans perturber les autres informations qui y figurent. La gestion indépendante de plusieurs zones de l'écran favorise la rapidité du jeu et augmente son intérêt.

Figure 8.39 : Le Ratissage des Dés

Additions et améliorations suggérées

Il y a bien des façons d'augmenter et d'améliorer les possibilités de ce jeu. Voici quelques suggestions.

- Empêcher qu'un joueur dans un excès d'enthousiasme ne déclenche par inadvertance le premier lancement de dés de la partie suivante, en appuyant plusieurs fois sur la barre d'espacement. On peut le faire en insérant

  repeat GET X$ until (X$ = "")

  en un endroit adéquat du programme. Il faudra décider si l'on désire empêcher un premier lancement intempestif de n'importe quelle partie, ou seulement de la partie du joueur suivant.

- Etudier l'introduction dans le jeu des possibilités complètes de pari telles qu'elles existent dans un casino. Cela impliquera que tout joueur (et pas seulement celui qui a les dés) pourra placer un enjeu ou retirer ses gains entre deux lancements de dés.

- Envisager un système de comptabilité différent, dans lequel les joueurs recevraient des jetons. Ce serait le total des jetons détenus par chacun qui serait affiché au lieu des gains et des pertes. (Ce système pourrait faciliter la mise en place du système complet de paris.) Quand les joueurs quittent la partie, leurs jetons seraient présentés à la caisse et comparés à ceux qu'ils ont achetés au début puis en cours de jeu. Un décompte final serait fait.

- Ajouter d'autres éléments statistiques comme par exemple, la meilleure série tous joueurs confondus (et pas seulement celle du joueur en cours), le nombre de lancements maximum avant obtention d'un résultat (pour tous les joueurs et pour le joueur en cours), les gains et les pertes les plus importantes, ou tout autre information à laquelle vous pourriez penser.

- Ajouter des possibilités de pari et en particulier les stratégies « automatiques » telles que « doubler ». (La plupart de ces stratégies sont expliquées dans les livres qui traitent des jeux de casino.)

- Afficher la valeur finale des statistiques de chaque joueur avant de passer au joueur suivant.

Si vous jouez à Craps pendant un certain temps, il vous viendra probablement bien d'autres idées.

En résumé

Craps est un jeu sur ordinateur qui a les caractéristiques du jeu « populaire » pratiqué dans les casinos. Le programme qui réalise Craps, illustre plusieurs points qui n'avaient pas été soulignés dans les commentaires concernant les autres jeux de ce livre.

Plusieurs éléments du jeu sont affectés à des zones de l'écran, gérées de façon indépendante grâce à un contrôle de position du curseur.

La rapidité et la continuité de l'action sont assurées par l'entrée mono-caractère, la disparition automatique au bout d'un temps précis de certains éléments affichés sur l'écran et les changements d'aspect de l'écran à chaque entrée au clavier.

Le passage des dés d'un joueur au suivant est un exemple simple de commutation de contexte, concept important dans les systèmes en temps partagé et autres programmes travaillant en multitraitement.

Le laps de temps imprévisible pendant lequel le programme attend la frappe d'une touche par le joueur, est mis à profit pour obtenir des valeurs de dés tout à fait aléatoires. Des appels répétés à la fonction RND sont effectués à l'intérieur de la boucle qui sert à l'attente du caractère en entrée.

De nombreuses améliorations peuvent être apportées à ce jeu, notamment en lui donnant des possibilités de pari supplémentaires et en perfectionnant les règles comptables.

CHAPITRE 9

La Vie Extra-Terrestre

La Vie Extra-Terrestre est un jeu composite dérivé de deux autres jeux : La Rencontre des Extra-Terrestres, jeu simple à caractère graphique imaginé par l'auteur et le Jeu de la Vie, jeu très connu, également à caractère graphique et souvent programmé pour des ordinateurs individuels. La Vie Extra-Terrestre résulte donc de la « greffe » du Jeu de la Vie sur la Rencontre des Extra-Terrestres.

La Rencontre des Extra-Terrestres

Le but de la Rencontre des Extra-Terrestres est la préparation de messages destinés à des créatures étrangères vivant à de nombreuses années-lumière de la Terre. Les messages que vous créez se composent de zéros et de uns qui correspondent au codage d'une image. Le programme de la Rencontre des Extra-Terrestres vous assiste dans la création de cette image et se charge du codage de celle-ci. La conception de l'image est de votre ressort.

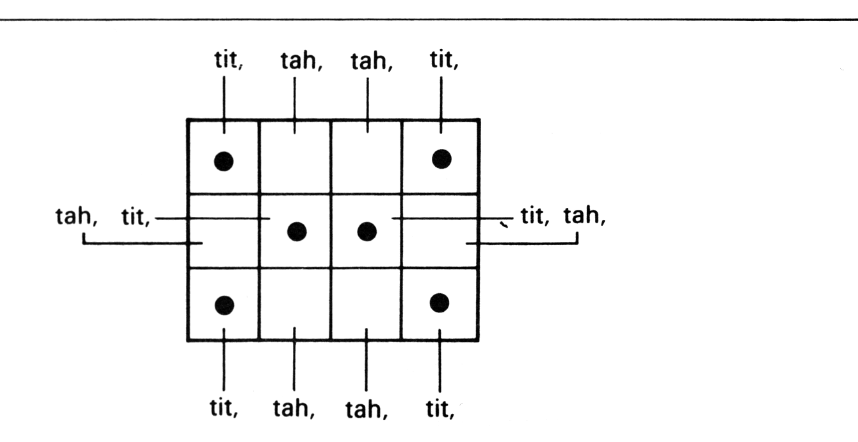
Dans *Intelligent Life in the Universe*, I. S. Shklovskii et Carl Sagan démontrent que notre technologie radio permet la transmission de suites de « zéros » et de « uns » de façon fiable à des distances interstellaires. Par zéros et uns, en matière de communication radio, nous voulons dire des points et des traits ou aussi, des « tits » et des « tahs » comme ceux utilisés dans le code Morse. La figure 9.1 montre comment on peut coder une matrice simple de 3 × 4 points par une suite de 12 tits et tahs.

Malheureusement, le codage repose sur le fait que celui qui reçoit sait ou ne sait pas qu'il s'agit d'une image réalisée par une matrice de 3 × 4 points. La figure 9.2 montre trois autres interprétations possibles en deux dimensions de cette même suite de « tits » et de « tahs ». La figure 9.3 montre l'une des trois interprétations possibles en trois dimensions.

Etant donné que les extra-terrestes, auxquels sont destinés nos messages, ne peuvent connaître nos intentions, nous devons minimiser le nombre des interprétations possibles. Nous pouvons le faire en émettant des suites de N zéros et uns (N bits, en somme) où N est le produit de deux nombres premiers *. Si N est le produit de deux

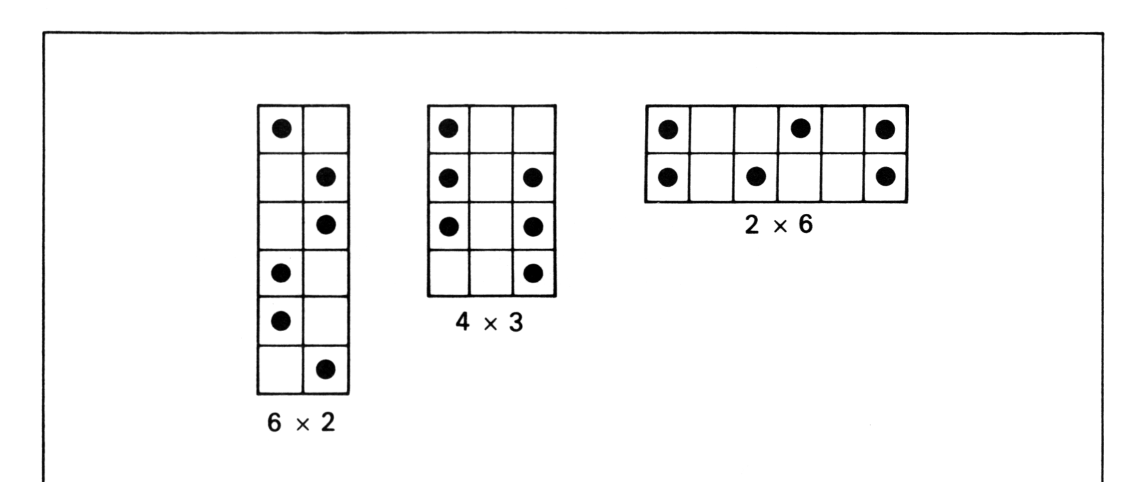
---

* Par définition, un nombre premier est un entier positif qui n'est pas le produit de deux autres entiers positifs. Par exemple, 2, 3, 5 et 7 sont premiers, mais 9 n'est pas premier, car 9 = 3 × 3. Par définition également, 1 n'est pas un nombre premier.



La suite de douze « tits » et « tahs » correspond aux points et aux blancs d'une image matricielle de 3 × 4. Cette séquence représente une transmission par radio composée de deux signaux différents qu'on peut distinguer comme le trait et le point du code Morse. Le message est émis plusieurs fois de façon à assurer la réception correcte des douze « tits » et « tahs ».

Figure 9.1 : Transmission d'une Image Simple

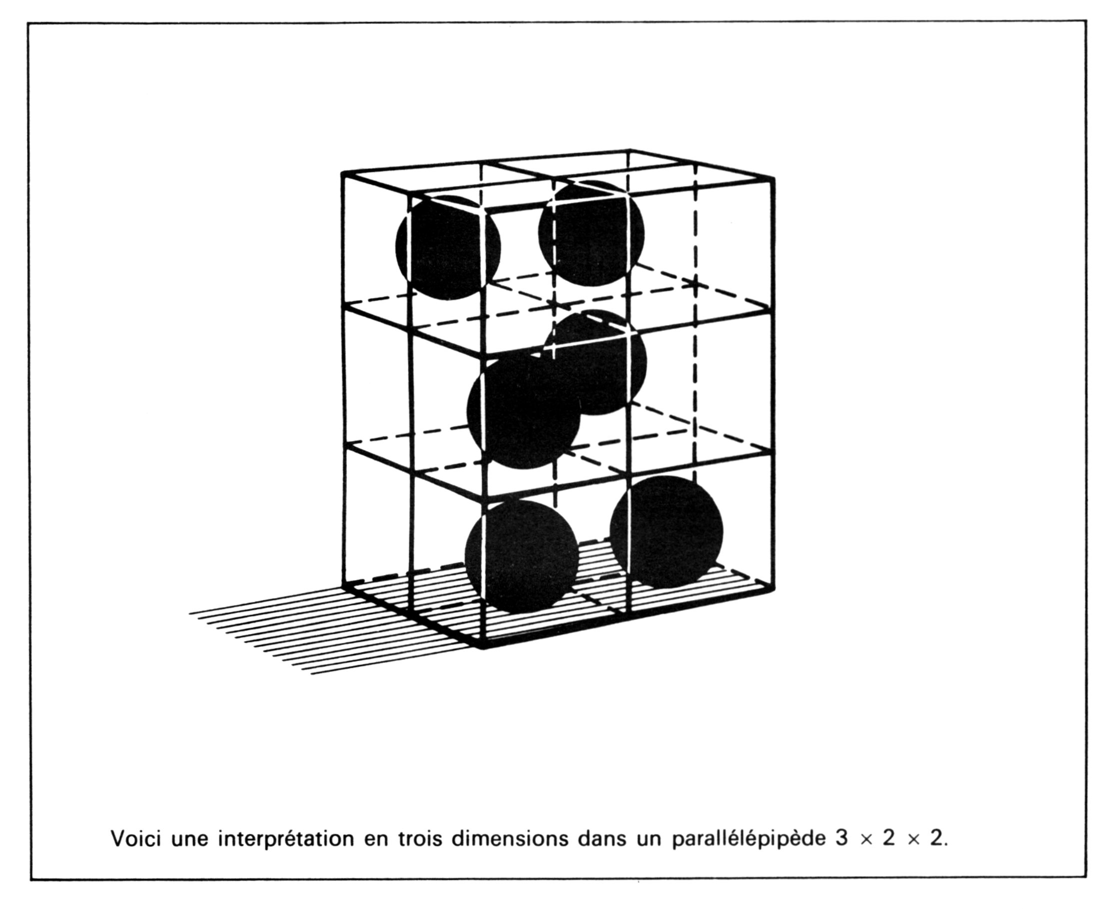


Voici trois autres interprétations de l'image de la figure 9.1. Si à la réception du message on ne sait pas qu'il représente une matrice 3 × 4, chacune de ces autres possibilités devra être envisagée.

Figure 9.2 : Autres Interprétations de l'Image Simple

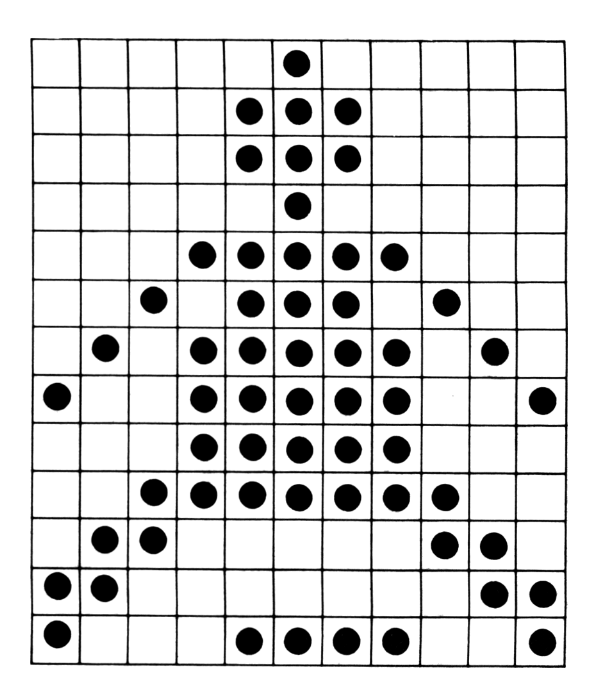
nombres premiers, il n'y a que deux interprétations matricielles d'une séquence de N bits.

Une image matricielle sophistiquée de 29 × 19 points imaginée en 1961 par Frank Drake, se trouve dans le livre déjà cité de Shklovskii et Sagan. La figure 9.4 montre une image matricielle de 13 × 11, dérivée de cette image 29 × 19. Elle représente l'auto-portrait d'un extra-terrestre et comprend une série de quatre points, grâce auxquels on pense que l'extra-terrestre pourra s'identifier lors des communications futures. La figure 9.5 montre la seule autre interprétation possible de la séquence de 143 bits de la figure 9.4.



Voici une interprétation en trois dimensions dans un parallélépipède 3 × 2 × 2.

Figure 9.3 : Une Interprétation en Trois Dimensions de l'Image Simple



Voici un auto-portrait matriciel 13 × 11 exécuté par une créature extra-terrestre. La rangée de quatre points dans le bas est le nom de la créature. C'est-à-dire que, lors de communications ultérieures, l'extra-terrestre utilisera une rangée de quatre points pour indiquer son identité. L'image a été adaptée du « premier message » imaginé, par Frank Drake qui créa en 1961 une matrice 29 × 19 sophistiquée.

Envoyé tel quel, le message en dessus serait constitué par la série suivante de « uns » et de « zéros ».

00000100000000001110000000001110000
00000100000000011111000000101110100
01011111010100111110010001111111000
00111111100011000001101100000000011
10001111001.

Cette séquence de « tits » et de « tahs » où 0 pourrait être un « tit » et 1 être un « tah » serait envoyé un nombre de fois suffisant pour assurer la réception correcte des 143 « bits ». Alors le savant qui le reçoit a simplement à deviner que ce message est un dessin selon une matrice 13 × 11.

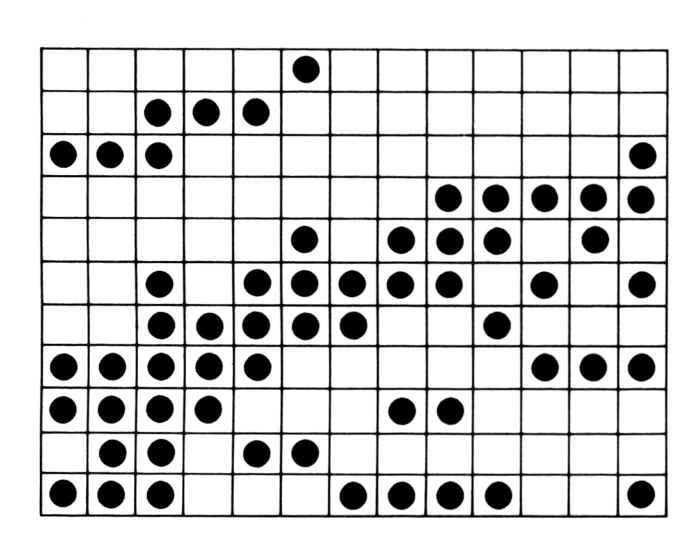
Figure 9.4 : Premier Message Provenant d'un Extra-Terrestre

Vous pouvez désirer échanger de tels messages avec d'autres amateurs de Rencontre avec les Extra-Terrestres, pour savoir si oui ou non, un autre être humain peut comprendre leur sens caché. Le programme présenté dans ce chapitre ne permet pas de tels échanges, mais des additions sont suggérées à cet effet en fin de chapitre.

Pour commencer, le programme de la Rencontre des Extra-Terrestres demande les dimensions du message envisagé. La figure 9.6 montre le dialogue correspondant. Le programme demande

LIGNES, COLONNES :

et vous répondez par deux nombres premiers séparés par une virgule. Si par hasard, les nombres en question étaient supérieurs au maximum



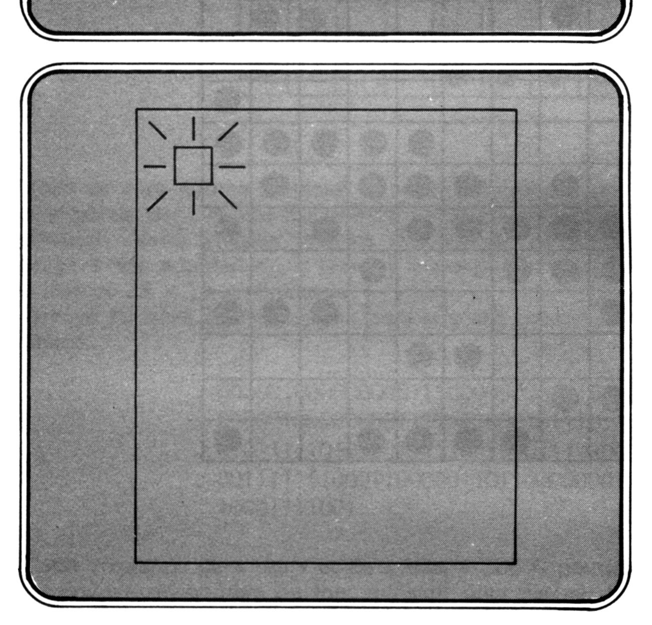
Voici l'image que l'on obtient en interprétant de façon erronée le message comme une matrice 11 × 13 au lieu d'une matrice 13 × 11.

Figure 9.5 : Mauvaise Interprétation du Message de l'Extra-Terrestre

autorisé ou si leur format n'était pas correct, le programme reposerait tout simplement la question. Cependant, si vos nombres ont des valeurs possibles sans être tous les deux premiers, le programme demandera

ACCORD POUR NON-PREMIERS ?

Si vous n'avez pas l'intention d'utiliser des nombres qui ne soient pas premiers, appuyez alors sur la touche " N ", et le programme vous redemandera les dimensions. Appuyer sur une autre touche amènerait le programme à utiliser les nombres composés, qui ont été indiqués. (Il y a une autre façon de répondre à la question du programme concernant les dimensions de l'image. Nous la verrons bientôt.)



Le programme demande les dimensions de l'image, et le joueur répond 13 par 11.

Le programme trace un cadre autour d'une zone de l'écran faisant 13 × 11 et fait apparaître un « curseur » clignotant dans le coin supérieur gauche.

Ceci est le dialogue initial de la Rencontre avec les Extra-Terrestres.

Figure 9.6 : Rencontre des Extra-Terrestres *(Début)*

Comme nous le voyons en figure 9.6, le programme, dès que les dimensions ont été décidées, trace un cadre autour de la zone réservée au dessin, et fait clignoter un curseur dans le coin supérieur gauche de cette zone. Le programme attend alors l'entrée d'une commande mono-caractère. Excepté les commandes " E " (exit, sortie) et " L " (Game of Life, Jeu de la Vie) qui seront décrites ultérieurement, les commandes que vous utiliserez sont faites en premier lieu pour créer ou supprimer des points, et ensuite pour déplacer le curseur d'image.

Chaque commande est exécutée en deux phases. Dans la première phase, une des trois actions suivantes a lieu :

— Un point est mis en place à la position occupée par le curseur.

— Le point éventuellement présent à la position du curseur est effacé.

— Aucun changement n'est apporté à la position du dessin, qu'il y ait un point ou un blanc, là où se trouve le curseur.

Dans la deuxième phase, le curseur se déplace dans l'une des quatre directions suivantes : vers le haut, vers le bas, vers la droite ou vers la gauche. Le déplacement du curseur se fait vers la position voisine dans la direction indiquée, mais avec effet « d'enroulement ». (Le terme « d'enroulement » est utilisé dans les cas où la « fin » et le « commencement » sont considérés comme des positions adjacentes.) C'est ainsi qu'un ordre de déplacement vers la droite, alors que le curseur se trouve déjà sur la colonne la plus à droite, a pour effet de placer le curseur d'image à la position la plus à gauche sur la même ligne.

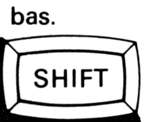
La figure 9.7 montre les commandes et la relation liant les commandes de dessin aux touches du clavier numérique. (Si votre ordinateur n'a pas de clavier numérique, vous serez peut-être amené à choisir d'autres caractères. Nous verrons comment on peut le faire lorsque nous étudierons le programme de la Vie Extra-Terrestre.) De manière à faciliter la mémorisation de ces nombreuses commandes, la méthode suivante a été utilisée :

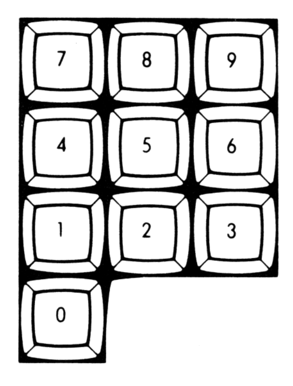
— Quatre touches ont été choisies pour commander le déplacement du curseur dans les quatre directions. En appuyant sur la touche SHIFT en même temps que sur l'une de ces quatre touches, on créera un point (à la position où se trouve le curseur), puis le mouvement indiqué se produira. Si l'on n'appuie pas sur SHIFT, seul le mouvement indiqué aura lieu sans changement de l'image.

— Les quatre touches de commande du curseur d'image ont été choisies aux quatre « coins » du clavier numérique. La touche

Commandes de Tracé

8* Curseur vers le haut
2* Curseur vers le bas
4* Curseur vers la gauche
6* Curseur vers la droite
0 Effacer le point, puis curseur vers la gauche
7 Effacer le point, puis curseur vers le haut
5 Effacer le point, puis curseur vers la droite
1 Effacer le point, puis curseur vers le bas.





* Avec SHIFT crée un point, puis déplace le « curseur » comme indiqué. Sans SHIFT déplace simplement le curseur.

Clavier Numérique

Commandes Diverses

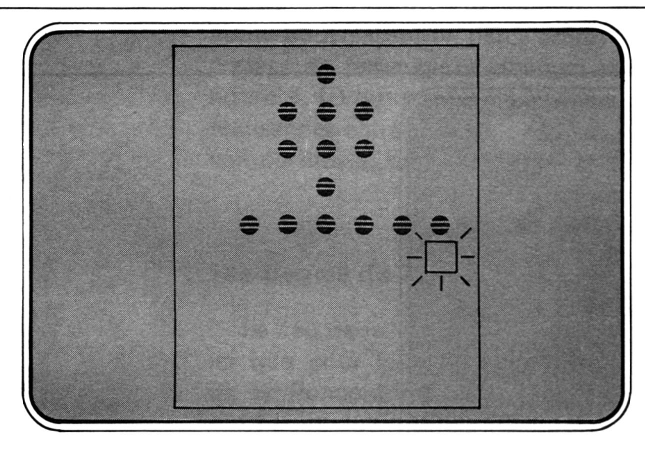
E Sortie (Exit). Transcrit l'image en une suite de uns et de zéros. (Prévue pour l'utilisation avec un stockage externe. Dans la version actuelle la suite binaire est seulement affichée sur l'écran.
L Vie (Life). Sera décrite ultérieurement.

Voici quelles sont les commandes que l'on peut envoyer au programme de la Rencontre avec les Extra-Terrestres quand le curseur clignotant est présent. Chacune d'elles ne comporte qu'un seul caractère.

Figure 9.7 : Les Commandes de la Rencontre des Extra-Terrestres

" vers le haut " est dans le coin du haut, la touche " vers le bas " occupe le coin du bas et celles qui permettent d'aller " à droite " et " à gauche ", se trouvent dans les « coins » correspondants.

Le premier point illustre « l'orthogonalité », terme emprunté à la théorie des espaces vectoriels. Plus simplement, les deux fonctions de sélection de l'action et de sélection du déplacement ont été affectées à deux « axes de coordonnées », et n'importe quelle action peut être combinée avec n'importe quel mouvement. Malheureusement, « l'axe » que nous avons choisi pour le choix de l'action ne possède que deux « points » : « La touche SHIFT est enfoncée » et « la touche SHIFT n'est pas enfoncée ». Aussi, n'est-il pas possible d'indiquer la troisième action (suppression d'un point) en appuyant sur une autre touche tout en utilisant une des quatre touches de direction. (Sur quelques ordinateurs individuels, un axe à trois valeurs est possible grâce aux touches SHIFT et CTRL. Les trois valeurs



$6,6,6,6,6,6^S,2,4^S,4^S,2^S,6^S,6^S,$
$2^S,4,4^S,4,2,6^S,6^S,6^S,6^S,6^S,2^S$

A partir de la position indiquée en figure 9.6, le joueur a commencé à tracer l'image de la figure 9.4. Sur la droite de l'écran on voit la suite de commandes mono-caractères que le joueur a émises. Un exposant « S » veut dire « avec la touche SHIFT ».

La séquence commence par cinq 6 de façon à amener le curseur au milieu de la ligne supérieure. Alors un 6^S entraîne la création d'un point puis le déplacement du curseur d'image d'une position vers la droite.

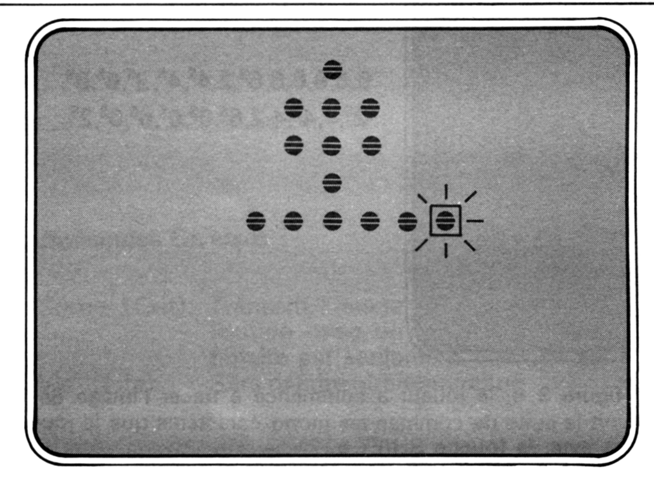
Un 2 amène le curseur à la position correspondant au point situé à l'extrême droite sur la deuxième ligne. Deux 4^S suivis d'un 2^S créent la deuxième ligne de l'image et placent le curseur à la position du point situé à l'extrême-gauche de la troisième ligne. Les commandes suivantes continuent ces aller et retour systématiques, mais vous remarquerez qu'une erreur a été commise.

Figure 9.8 : On Commence à Dessiner l'Extra-Terrestre

correspondent alors à « SHIFT enfoncée », « CTRL enfoncée » et « aucune des deux enfoncée ». Si vous possédez un tel système, vous serez peut-être tenté de modifier les codes de commande pour tirer parti de cette situation.)

Maintenant que la description des commandes qui permettent de dessiner est achevée, regardons un exemple d'utilisation. A la figure 9.8, vous avez commencé à dessiner l'image de la figure 9.4. Partant de la position indiquée à la figure 9.6, vous avez utilisé la suite de commandes montrée en figure 9.8, pour réaliser un « balayage » aller et retour achevant le dessin de chaque ligne de points avant de passer à la suivante.

Par malchance, votre enthousiasme vous a amené à dessiner un point de trop à la cinquième ligne de l'image. Pour faire disparaître ce point, vous devez vous servir de la commande qui déplace le curseur d'une position vers le haut comme le montre la figure 9.9. Maintenant que le curseur se trouve placé sur le point superflu, l'une quelconque des quatre commandes « effacer le point et déplacer le curseur », peut être utilisée pour éliminer le gêneur. La commande



Partant de la dernière position de la figure 9.8 le joueur a appuyé sur 8 pour remettre le curseur d'image sur le dernier point dessiné. L'apparence du curseur diffère suivant qu'à la position où il se trouve, il y a un point ou non.

Maintenant le joueur peut faire disparaître le point qui se trouve au même endroit que le curseur avec l'une des quatre commandes 0, 7, 5 ou 1. Chacune de ces commandes laissera le curseur dans une position différente.

Figure 9.9 : Suppression d'un Point

« effacer et aller vers le bas » est celle qui cadre le mieux avec la méthode de dessin que vous avez suivie jusqu'à présent.

La figure 9.10 montre comment obtenir la version codée en binaire de l'image. Chaque fois que le curseur d'image clignote, l'entrée de la commande « E » entraîne l'affichage de la forme codée de l'image. Les zéros correspondent aux blancs, les uns aux points de l'image.

La figure 9.10 illustre une autre caractéristique du programme de la Vie Extra-Terrestre. Vous venez d'examiner l'image sous sa forme codée et vous avez appuyé sur une touche quelconque (sauf la lettre « Q ») de façon à faire poser par le programme la question

LIGNES, COLONNES :

Mais au lieu de préciser de nouvelles dimensions, vous avez appuyé sur RETURN, ce qui fait reparaître sur l'écran l'image sur laquelle vous venez de travailler.

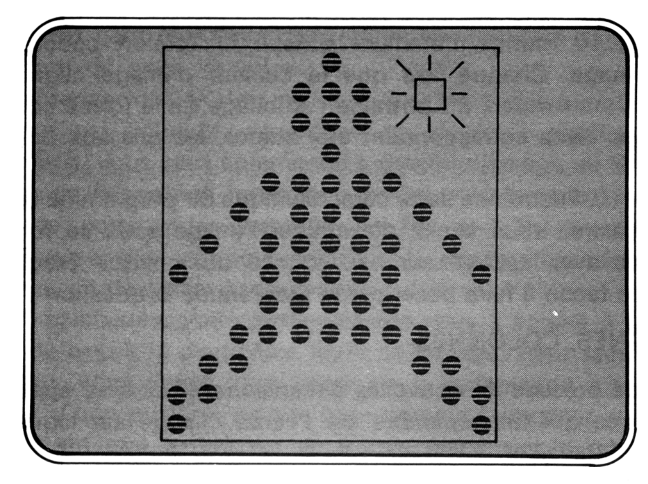
C'est maintenant que vous désireriez sauvegarder la version codée de l'image, sur cassette disque ou papier, de façon à la transmettre à un autre joueur de Rencontre des Extra-Terrestres. Vous aimeriez aussi pouvoir entrer dans votre ordinateur individuel, des images codées en provenance d'autres joueurs. Les actions montrées à la figure 9.10 ont pour origine des « souches » dont le but final est de réaliser des entrées/sorties d'images codées, soit de façon « manuelle » soit au moyen de disques ou de cassettes.

Les Règles du Jeu de la Vie

Le Jeu de la Vie est un jeu en lui-même, mais nous n'en parlerons ici que pour les possibilités supplémentaires qu'il apporte au jeu de la Rencontre avec les Extra-Terrestres, en matière de dessin.

Chaque fois que le curseur d'image clignote, on peut entrer la commande "L". Cette commande entraîne la transformation de l'image suivant les règles du Jeu de la Vie, par le programme. Nous allons énoncer ces règles sans étudier leur origine, qui viennent de l'observation du comportement de populations d'organismes.

La transformation de l'image selon les règles du Jeu de la Vie, implique le calcul du nombre des voisins de chaque position. La figure 9.11 montre ce que l'on entend par positions « voisines ». Les positions voisines d'une position donnée sont celles qui la « touchent », suivant les axes verticaux et horizontaux et suivant les diagonales à 45 degrés. En général, une position possède huit, cinq ou trois positions voisines suivant l'endroit où elle se situe par



L'image de l'extra-terrestre est achevée, et le curseur d'image a été déplacé vers le coin supérieur droit.

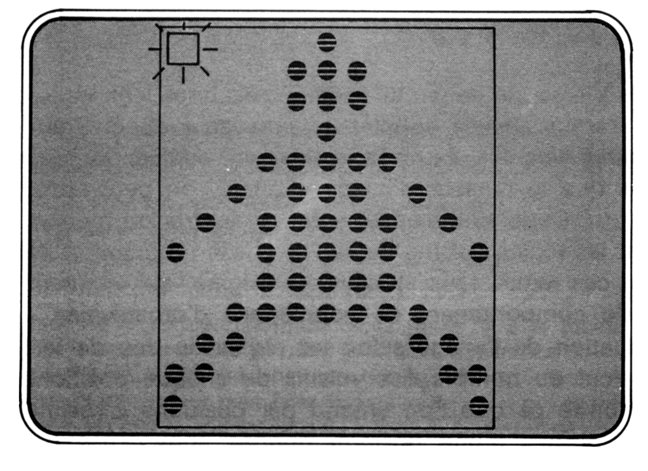
Le joueur entre alors la commande « E ».

```
000001000000000011100000000111000000000010
000000011111000001011101000101111101 0100
111110010001111100000111111100011000001 1
011000000001110001111001
```

L'écran est effacé et le codage de l'image en une suite de zéros et de uns est affiché. Le joueur appuie sur RETURN.

```
LIGNES, COLONNES :
```

Le programme réclame les dimensions d'un autre cadre. Le joueur appuie sur RETURN pour faire reparaître l'image précédente.



Le programme est prêt à accepter de nouvelles commandes.

Figure 9.10 : Codage de l'Image

rapport aux « bords » de la zone. Quant au nombre des *voisins* d'une position donnée, c'est celui des points qui occupent une position voisine. Par exemple à la figure 9.10, la position du point au sommet de la tête de l'extra-terrestre (position (0, 5)) a trois voisins qui sont les trois points du dessous (aux positions (1, 4), (1, 5) et (1, 6)). Les positions situées à gauche et à droite du point du sommet ont également trois voisins. Les voisins de la position (0, 4) sont en (1, 4), (0, 5) et (1, 5) et les voisins de la position (0, 6) sont en (0, 5), (1, 5) et (1, 6). La position qui comporte un point et se trouve immédiatement sous le point du sommet (position (1, 5)), au milieu de la tête de l'extra-terrestre, a six voisins : les points aux positions (0, 5), (1, 4), (1, 6), (2, 4), (2, 5) et (2, 6). Les points aux extrémités des bras n'ont qu'un seul voisin chacun.

La figure 9.12 présente les règles de transformation du Jeu de la Vie. La figure 9.13 en montre des exemples d'application. Nous ne ferons pas d'étude détaillée des situations extrêmement diverses qui peuvent se présenter dans le Jeu de la Vie. Cela a été fait par ailleurs et ne concerne notre propos que de façon marginale.

Voyons maintenant quel est l'effet de la commande "L" sur l'auto-portrait de notre extra-terrestre. La figure 9.14 montre les étapes par lesquelles passe le portrait, avant d'atteindre une configuration stable. L'extra-terrestre a-t-il voulu donner sur lui-même des renseignements supplémentaires en montrant comment se

(0, 0) (0, 1) (0, 2)... (0, HC) (1, 0) (1, 1) (1, 2)... (1, HC) (2, 0) (2, 1) (2, 2)... (HL, 0) (HL, 1)...(HL, HC)	(1, 1) a huit positions voisines : (0, 0), (0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1), (2, 2). (1, 0) a cinq positions voisines : (0, 0), (0, 1), (1, 1), (2, 0), (2, 1). (0, 0) a trois positions voisines seulement : (0, 1), (1, 0), (1, 1).

Les positions de la zone image ont été repérées par leurs numéros de lignes et de colonnes de façon à pouvoir illustrer l'idée de « positions voisines ». En général, une position « loin du bord » a huit positions voisines. Une position « le long du bord » mais éloignée d'un « coin » a cinq positions voisines. Les positions de « coin » ont trois positions voisines. Dans le cas particulier où la zone image a seulement une ligne ou seulement une colonne, le nombre des positions voisines est soit un soit deux.

Figure 9.11 : Les Positions Voisines

transforme son auto-portrait, lorsqu'on le soumet de façon répétée aux règles du Jeu de la Vie ? Est-ce la configuration finale stable ou bien est-ce le nombre de transformations nécessaires pour y arriver, qui a une signification particulière ? Peut-être les stades intermédiaires ont-ils eux aussi leur mot à dire. Ces possibilités montrent la dimension nouvelle donnée à la Rencontre des Extra-Terrestres par le Jeu de la Vie.

Si vous avez déjà utilisé d'autres programmes de Jeu de la Vie, vous savez que les capacités graphiques de la Rencontre des Extra-Terrestres et la possibilité de modifier l'image entre deux transformations quelconques, ne sont pas habituellement disponibles sur les programmes de Jeu de la Vie. De plus, le stockage des images sous forme binaire sur des mémoires externes, comme nous le suggérons à la fin de ce chapitre, fournirait une possibilité qui n'est généralement pas à la disposition des amateurs de Jeu de la Vie.

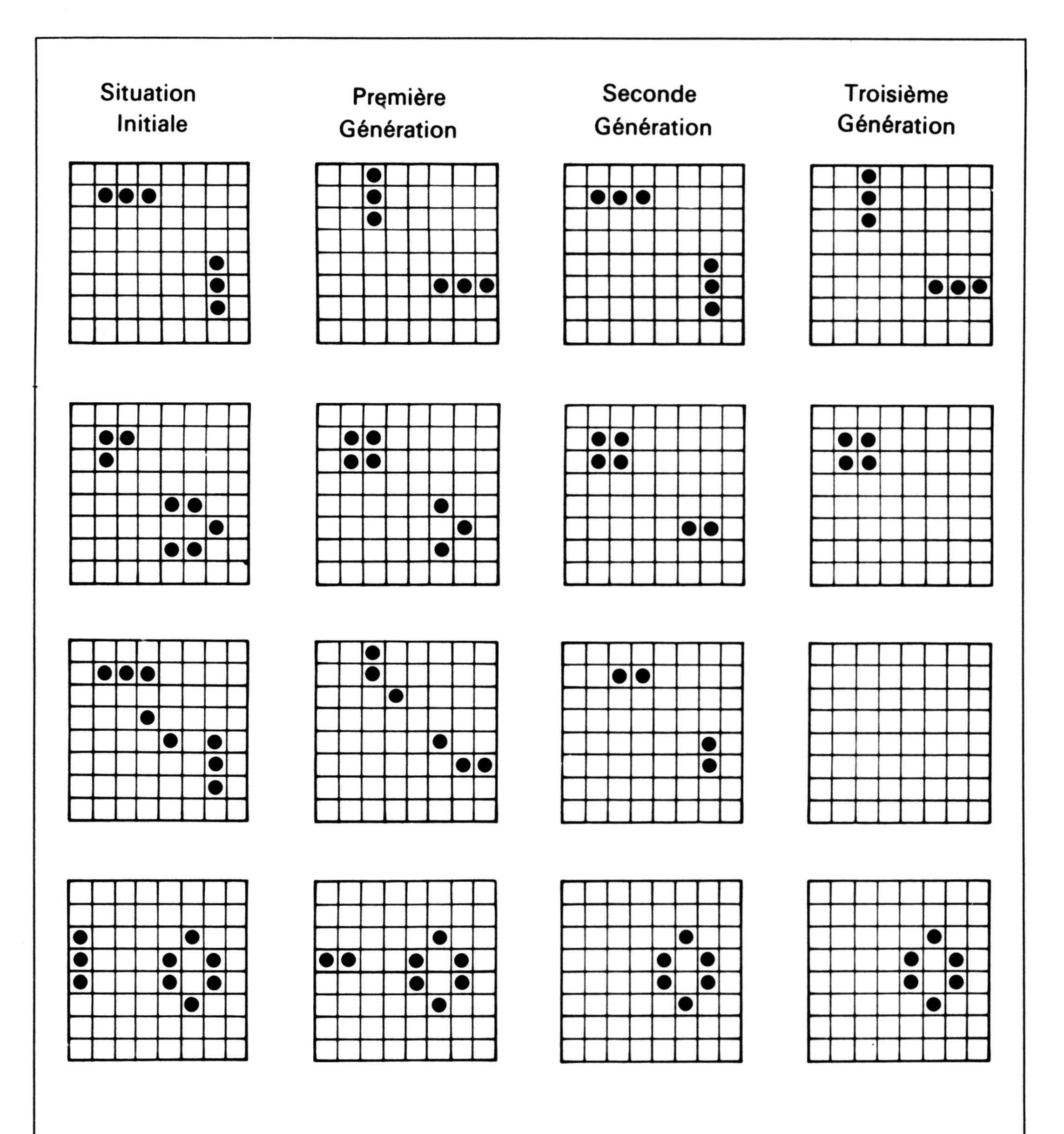
Le programme de la Vie Extra-Terrestre

Maintenant que nous avons décrit le jeu proprement dit, voyons le programme qui le réalise. Ce programme qui est présenté de la figure 9.15 à la figure 9.44, ressemble en bien des points aux autres programmes de ce livre. En le décrivant, nous verrons tout particulièrement la zone réservée à l'image et les commandes. En ce qui concerne la zone image, nous étudierons la programmation du cadre qui entoure la zone, celle des points qui constituent l'image et enfin celle du curseur d'image. Pour les commandes, nous décrirons comment elles sont codées et comment il est possible d'en ajouter facilement de nouvelles.

Nombre de voisins	Nouveau contenu
Inférieur à 2	Pas de point
2	Le contenu reste le même
3	Un point
Supérieur à 2	Pas de point

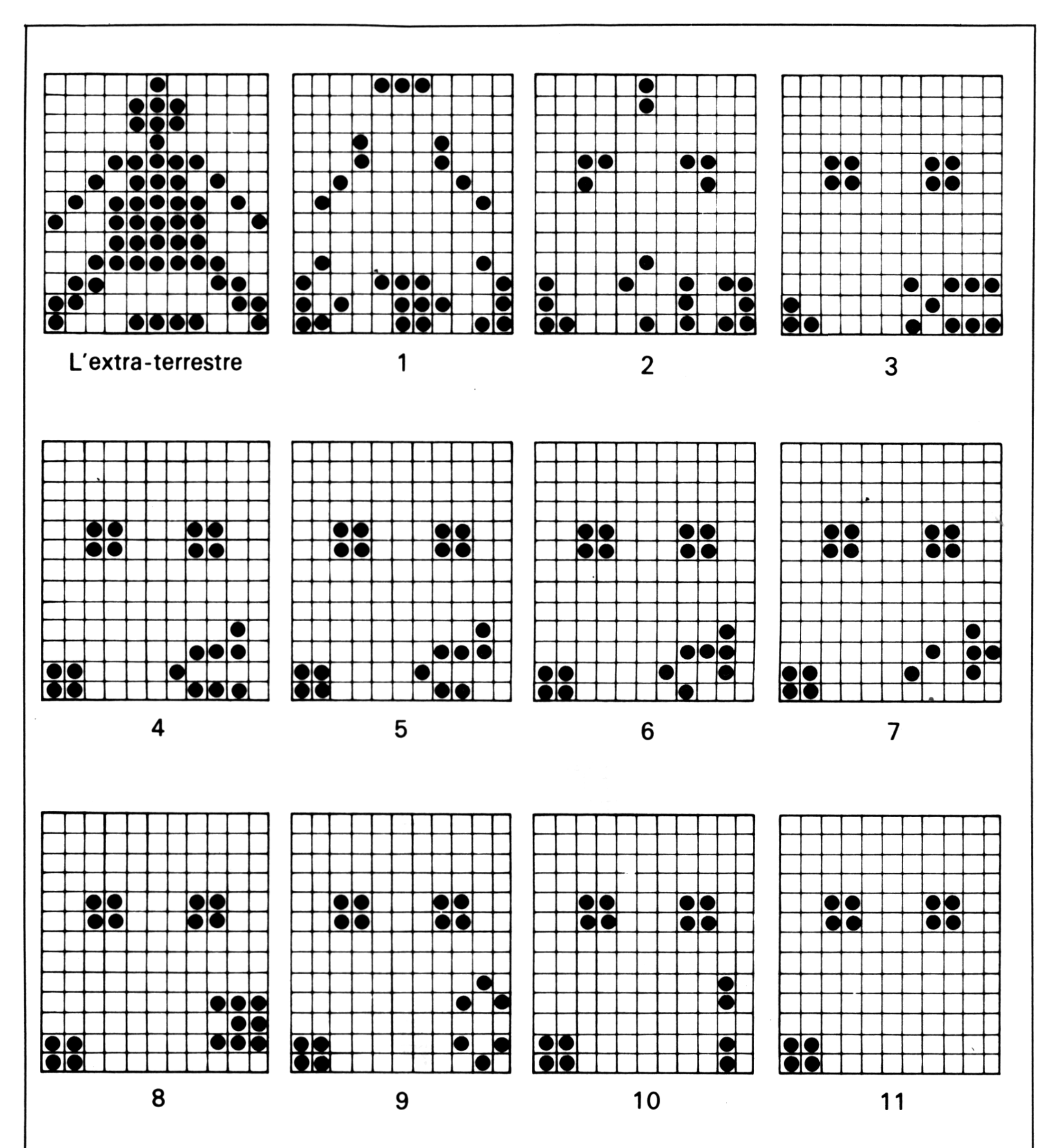
Voici les règles appliquées pour la transformation de l'image lorsqu'on entre la commande "L" (Jeu de la Vie). Le programme commence par calculer le nombre de voisins pour chacune des positions de la zone image. Le nouveau contenu est fonction de l'ancien contenu et du nombre de voisins conformément aux règles ci-dessus.

Figure 9.12 : Les Règles du Jeu de la Vie



Ces quelques exemples montrent ce que deviennent les configurations de points lorsqu'on leur applique de façon répétitive, les règles du Jeu de la Vie. Certaines se volatilisent, d'autres deviennent stables, d'autres encore repassent indéfiniment par la même succession d'états.

***Figure 9.13 : Le Jeu de la Vie** (Exemples)*



En appliquant onze fois de suite les règles du Jeu de la Vie au portrait de l'Extra-Terrestre, on obtient une configuration stable composée de trois amas carrés comprenant chacun quatre points.

Figure 9.14 : ***Le Jeu de la Vie*** *(Transformations)*

La zone image

La zone image comprend trois éléments essentiels :

— Le cadre.

— Les points d'image.

— Le curseur d'image.

Le cadre est tracé une seule fois, par les routines appelées à partir de *encadrer* (fig. 9.16). La routine *centrer* (fig. 9.25) calcule la position du coin supérieur gauche, de telle sorte que le cadre soit centré sur l'écran. La routine *dessinercadre* (fig. 9.26) trace un cadre rectangulaire délimitant la zone image sur l'écran. L'appel lancé par *encadrer* à *dessinercadre* est précédé d'un appel à *effacécran,*

```
# La Rencontre des Extra-Terrestres et le Jeu de la Vie
   GOSUB init                      # initialiser tableaux et chaînes
   repeat {
       GOSUB encadrer              # tracer un nouvel encadrement d'image
       repeat {
           GOSUB cmd/dessin        # convertir un caractère entré en deux nombres
           IF C1 <> 0 THEN         # C1, C2 ; C1 = 0 veut dire terminé
               ON C1 GOSUB         # exécution d'une commande de dessin
                   deplacement     # déplacer curseur d'image
                   creermarque     # faire une marque (et déplacer le curseur)
                   tuermarque      # effacer une marque (et déplacer le curseur)
                   cmd/div         # autres commandes
           } until (C1 = 0)
       GOSUB binaire               # écrire l'image sous forme de uns et de zéros
       GOSUB uncar                 # signal pour commencer un autre cadre
       } until (X$ = "Q")          # Q = sortie
   END

 100  GOSUB1190
 110  GOSUB160
 120  GOSUB700:IFC1=0THEN140
 130  ONC1GOSUB720,730,740,750:GOTO120
 140  GOSUB560:GOSUB990:IFX$<>"Q"THEN110
 150  END
```

Ceci est la structure principale du programme de la Vie Extra-Terrestre, programme qui associe les règles du Jeu de la Vie à celles du Jeu de la Rencontre des Extra-Terrestres.

La structure ci-dessus est semblable à celle des autres routines principales de ce livre.

Figure 9.15 : La Vie Extra-Terrestre

de sorte que l'écran soit propre au moment où *dessinercadre* est appelé.

Les points de l'image sont le reflet du tableau D. La routine *point/blanc* (fig. 9.27) place un point à chaque position définie par la ligne LL et la colonne CC pour laquelle on a D(LL, CC) <> 0; elle place un blanc aux positions (LL, CC) pour lesquelles D(LL, CC) = 0. L'ordre dans lequel ceci est fait, a une influence sur l'aspect de l'écran. Par exemple, quand on se sert de la commande " L" pour obtenir une transformation de l'image selon les règles du Jeu de la Vie, l'image est modifiée ligne par ligne.

```
# Dessiner le cadre de l'image

encadrer  GOSUB effacecran           # démarrer sur écran propre
          GOSUB premiers             # demander les dimensions de l'image
          IF PX = - 1 THEN {         # PX = - 1 pour charger une configuration
                                     stockée
               GOSUB chargerimage    # lire la configuration
               GOSUB choix           # lire P1, P2, HL, HC
               }
          else {
               HL = P1 - 1           # numéro dernière ligne
               HC = P2 - 1           # numéro dernière colonne
               GOSUB pointazero      # mettre à zéro la partie HL × HC du
                                     tableau D
               }
          GOSUB centrer              # calculer LZ et CZ pour coin supérieur
                                     gauche
          GOSUB effacecran           # effacer l'écran
          GOSUB dessinercadre        # dessiner un cadre
          GOSUB point/blanc          # dessiner points ou blancs suivant tableau D
          CC = 0 : LL = 0            # positionner le curseur
          GOSUB nouveaucurseur
          RETURN

160 GOSUB980:GOSUB200:IFPX<>-1THEN180
170 GOSUB540:GOSUB550:GOTO190
180 HL=P1-1:HC=P2-1:GOSUB600
190 GOSUB610:GOSUB980:GOSUB620:GOSUB660:CC=0:LL=0:GOSUB840:RETURN
```

La routine « encadrer » saisit les dimensions du cadre et dessine celui-ci au centre de l'écran. L'une des options permet « l'entrée groupée », c'est-à-dire sous forme de suite binaire. Cette entrée binaire peut se faire au clavier ou à partir d'une mémoire externe sur laquelle l'image aura été au préalable sauvegardée.

Figure 9.16 : Réalisation du Cadre

```
# Saisir les dimensions de l'image — retourne PX = – 1 si entrée groupée (binaire)

premiers  repeat {
              PRINT "LIGNES, COLONNES : ";     # demander les dimensions
              GOSUB entreechaine               # obtenir une réponse
              IF XX$ = " " THEN                # touche RETURN veut dire
                                                 entrée groupée binaire
                  { PX = – 1 : RETURN }
              else {
                  GOSUB saisirpremiers         # analyser la chaîne entrée
                  IF PX = 0 THEN               # erreur de syntaxe
                      XX = – 1
                  else {
                      GOSUB controlepremiers   # contrôler les nombres
                      IF XX = 1 THEN {         # si pas premier
                          GOSUB cava           # demander si ça va quand
                                                 même
                          IF OK = 1 THEN
                              XX = 0
                          }
                      }
                  }
              } until (XX = 0)
          P1 = PX : P2 = PY                    # PX et PY sont nombre de
                                                 lignes et de colonnes
          RETURN
```

```
200 PRINT"LIGNES,COLONNES:";:GOSUB1010:IFXX$<>""THEN220
210 PX=–1:RETURN
220 GOSUB310:IFPX<>0THEN240
230 XX=–1:GOTO200
240 GOSUB420:IFXX<>1THEN270
250 GOSUB290:IFOK<>1THEN270
260 XX=0
270 IFXX<>0THEN200
280 P1=PX:P2=PY:RETURN
```

La routine « premiers » demande au joueur quelles sont les dimensions du cadre. Pour le Jeu de la Rencontre des Extra-Terrestres, les dimensions devraient être des nombres premiers. Si ce ne sont pas des nombres premiers, la routine demande confirmation avant de les accepter.

Figure 9.17 : Les Dimensions de l'Image

Une autre solution consisterait à « blanchir » toute la zone image, puis à recopier les points. Un des avantages de cette façon de procéder (qu'on peut réaliser en modifiant la routine *point/blanc*), est la diminution, dans la majeure partie des cas, du temps requis pour le tracé de l'image. Le choix entre les deux méthodes est une affaire de convenance personnelle.

La routine *point/blanc* est, dans un premier temps, appelée à partir d'*encadrer* (fig. 9.16). Par la suite, elle n'est appelée qu'à partir de la routine *Vie* (fig. 9.44). Les routines, mises en œuvre par les commandes de dessin, affichent (ou effacent) à un instant donné, un point et un seul, par contrôle de la position du curseur, sans redessiner la totalité de l'image. Les routines d'affichage et d'effacement sont présentées en figure 9.32. Elles appellent les routines *créerpoint* et *tuerpoint* (fig. 9.27) qui assurent le travail effectif.

Le curseur d'image est totalement différent du curseur du tube vidéo qui est contrôlé par votre ordinateur individuel et qui vous guide pour la saisie des programmes et des données. Le curseur d'image est en réalité sous le contrôle du programme de la Vie Extra-Terrestre. Ce contrôle porte sur les points suivants :

— emplacement du curseur d'image
— choix du caractère représentant le curseur
— clignotement
— réaffichage du caractère adéquat lorsque le curseur change de place.

L'emplacement du curseur d'image est défini par les variables de ligne et de colonne LL et CC. Ces variables sont mises à zéro par les routines *encadrer* (fig. 9.16) et *Vie* (fig. 9.44), ce qui a pour effet de placer le curseur d'image dans le coin supérieur gauche de la zone. Les commandes de tracé déplacent le curseur à partir de cette position. Les routines présentées à la figure 9.33 modifient les coordonnées contenues dans les variables LL et CC. Le mouvement lui-même est le fait de la routine *entréecurseur* (fig. 9.36).

La routine *entréecurseur* est appelée à partir de *cmd/dessin* (fig. 9.28) pour obtenir du joueur l'entrée de la commande suivante. *Entréecurseur* choisit d'abord le caractère du curseur en appelant *nouveaucurseur* (fig. 9.32). Elle exécute alors une boucle de programme au cours de laquelle elle affiche tour à tour le caractère choisi et un blanc, en attendant une entrée au clavier. Cet affichage alterné constitue le « clignotement » du curseur. La fréquence du clignotement dépend de la variable FQ dont la valeur est fixée dans la routine *init* (fig. 9.38).

La routine *entréecurseur* présentée à la figure 9.36, utilise la variable TI propre au Pet pour régler le rythme du clignotement. C'est pour cette raison que la valeur de FQ est donnée en soixantième de seconde, unité de temps de l'horloge du Pet. Sur TRS-80, la boucle *repeat...until* est remplacée par une boucle FOR...NEXT qui a pour limites

1 TO FQ * TN

où TN a une valeur qu'on détermine empiriquement, de telle sorte que la bouche soit exécutée en FQ unités de temps. Le TRS-80 utilise l'instruction

X$ = INKEY$

tandis que pour le Pet, c'est

GET X$

La version Apple ne peut pas utiliser cette technique de saisie mono-caractère. En effet, l'instruction Apple

GET X$

fait clignoter le curseur vidéo, et met le système en attente jusqu'à ce que le joueur ait rentré un caractère. Aussi, la version Apple d'*entréecurseur* consiste-t-elle simplement en un appel à *curseur*

```
# Demander si ça va avec non-premiers — Renvoyer OK = 0 si non, OK = 1 si oui

çava  PRINT " NON-PREMIERS OK ? ";    # poser la question
      GOSUB uncar                     # obtenir la réponse
      IF X$ = " N " THEN
          OK = 0                      # " N " = non
      else
          OK = 1                      # toute autre réponse = oui
      RETURN

290  PRINT"NON-PREMIERS OK?";:GOSUB990:OK=1:IFX$="N" THENOK=0
300  RETURN
```

La routine « ça va » demande au joueur si les nombres non-premiers qui viennent d'être entrés sont réellement utilisables. Si le joueur a rentré par inadvertance un nombre non-premier, la réponse " N " indiquera au programme qu'il faut redemander les dimensions.

Figure 9.18 : Utilisation de Nombres Non-premiers *(demande de confirmation)*

```
# Extraire de XXS deux nombres (dimensions) — Si erreur PX = 0

saisirpremiers
    LL = LEN(XX$)                               # longueur chaîne entrée
    IF LL < 3 THEN                              # insuffisant
        erreur
    else {
        NP = 0                                  # on n'a rien pour l'instant

        FOR SX = 1 TO LL - 1
            X$ = MID$(XX$, SX, 1)               # caractère suivant de XX$

            GOSUB separateur                    # séparateur ?
            IF SP = 0 THEN                      # non
                NP = 1                          # trouvé un nombre dans la chaîne

            else                                # oui
                IF NP = 0 THEN                  # avant le premier nombre premier

                    IF X$ <> " " THEN           # les espaces sont autorisés

                        erreur
                else {
                    PX = VAL(LEFT$(XX$, SX - 1))  # premier nombre premier

                    PY = VAL(MID$(XX$, SX + 1))   # second nombre premier

                    RETURN
                    }
            NEXT SX
        erreur                                  # pas trouvé les deux
        }


310 LL=LEN(XX$):IFLL>=3THEN330
320 PX=0:RETURN
330 NP=0:FORSX=1TOLL-1:X$=MID$(XX$,SX,1):GOSUB400:IFSP<>0THEN350
340 NP=1:GOTO390
350 IFNP<>0THEN380
360 IFX$=""THEN390
370 PX=0:RETURN
380 PX=VAL(LEFT$(XX$,SX-1)):PY=VAL(MID$(XX$,SX+1)):RETURN
390 NEXTSX:PX=0:RETURN
```

La routine «saisirpremiers» extrait les deux dimensions de la chaîne XX$ retournée par la routine «entréechaine». Le format d'entrée est supposé être : nombre, séparateur, nombre. Le séparateur est soit une virgule soit un espace. Par exemple, «7, 5» et «7 5» sont valables.

Figure 9.19 : Analyse de la Chaîne Contenant les Dimensions

avec C et L paramétrés aux valeurs CZ + CC et LZ + LL, suivi de l'instruction

GET X$

Le dernier point à voir est le retour de l'écran à son contenu véritable (sans curseur). L'appel à *sanscurseur* (fig. 9.32), qui a lieu juste avant l'instruction RETURN marquant la fin d'*entréecurseur*, a pour effet d'afficher le caractère adéquat (tel qu'il est stocké dans le tableau D) à l'endroit où clignotait le curseur d'image.

Ceci termine notre étude de la zone image. Nous allons maintenant voir les commandes du programme de la Vie Extra-Terrestre.

Les commandes

Le programme de la Vie Extra-Terrestre accepte un certain nombre de commandes mono-caractères. Dans la présente partie, nous étudierons :

— le codage utilisé pour condenser la commande primaire et la commande secondaire en un seul nombre

— le mécanisme incorporé qui permet d'accepter de nouvelles commandes

— l'association aux commandes de caractères spécifiques.

La routine *cmd/dessin* de la figure 9.28 montre le codage utilisé pour les commandes. *Cmd/dessin* commence par appeler *entrée-curseur* pour en obtenir un caractère ASCII (unique), que *cmd/dessin*

```
# Recherche du separateur

separateur   IF X$ =" "OR X$ = "," THEN
                   SP = 1
             else
                   SP = 0
             RETURN

  400  SP=0:IFX$=" "ORX$=","THENSP=1
  410  RETURN
```

La routine « séparateur » regarde si le caractère est ou non un séparateur. Les caractères de séparation reconnus par cette routine sont la virgule et l'espace, mais d'autres pourraient y être ajoutés facilement.

Figure 9.20 : Identification des Séparateurs

utilise (d'une façon que nous allons bientôt décrire) pour accéder à l'élément de tableau C(CX) où se trouve le nombre contenant les éléments de la commande. Le calcul de C1 et C2 est basé sur le fait que le nombre est censé être de la forme

$$n_2 + 100 \times n_1$$

où n_2 est inférieur à 100. C2 prend la valeur n_2 et C1, de n_1. La compression de ces deux commandes numériques en un nombre unique a lieu dans la routine *init* (fig. 9.38 à 9.40).

Les commandes primaires et secondaires sont utilisées de la façon suivante. Tout d'abord, la valeur de C1 est utilisée dans le

```
# Contrôler PX et PY — retourner XX = – 1 si faux, 0 si premiers, 1 si non-premiers
controlepremiers  IF PX > ML OR PY > MC THEN    # hors limites
                        XX = – 1
                  else {
                        PP = PX : GOSUB testpremiers
                        IF PR = 0 THEN
                              XX = 1
                        else {
                              PP = PY : GOSUB testpremiers
                              IF PR = 0 THEN
                                    XX = 1
                              else
                                    XX = 0
                              }
                        }
                  RETURN

420 IFPX<=MLANDPY<=MCTHEN440
430 XX=-1:RETURN
440 PP=PX:GOSUB480:IFPR<>0THEN460
450 XX=1:RETURN
460 PP=PY:GOSUB480:XX=0:IFPR=0THENXX=1
470 RETURN
```

La routine « contrôlepremiers » vérifie les nombres contenus dans PX et PY et renvoie le résultat de cet examen dans la variable XX. XX est mis à – 1 si l'un ou l'autre des deux nombres dépasse le maximum prévu, à 0 si les deux nombres sont premiers, ou à 1 si l'un ou l'autre des deux nombres n'est pas premier. C'est la routine « testpremier » qui vérifie si les nombres sont premiers ou non. Elle est appelée une fois pour chacun des nombres.

Figure 9.21 : Les Dimensions sont-elles des Nombres Premiers ?

programme principal (fig. 9.15). Il y a trois codes de commande primaires :

— le code « exit »

— un code de dessin

— le code « divers ».

Si la commande primaire est « exit », l'information secondaire n'est pas utilisée. S'il s'agit d'un code de dessin, l'information secondaire sera utilisée dans *bougercurseur* (fig. 9.31). Quant au code « divers », il entraînera le traitement de l'information secondaire dans la routine *divers* (fig. 9.30).

La routine *divers* est l'endroit où de nouvelles commandes peuvent être ajoutées au programme. Le Jeu de la Vie fournit un bon exemple de cette possibilité. En effet, son incorporation au programme n'a eu lieu qu'après l'achèvement de la Rencontre des Extra-Terrestres. Pour incorporer le Jeu de la Vie, il a fallu :

— définir la commande " L " comme « numéro 1 des commandes diverses »

```
# Tester PP — retourner PR=1 si PP est premier, PR = 0 sinon

testpremier   IF PP = 2 OR PP = 3 OR PP = 5 THEN
                  retourpremier
              else IF PP < 7 OR PP is even THEN
                  retournonpremier
              else {
                  FOR I = 3 TO INT(SQR(PP)) STEP 2    # essayer les diviseurs
                                                        possibles
                      IF I divise PP THEN
                          retournonpremier
                      NEXT I
                  retourpremier
                  }

480  IFPP=2ORPP=3ORPP=5THENPR=1:RETURN
490  IFPP<7ORPP/2=INT(PP/2)THENPR=0:RETURN
500  FORI=3TOINT(SQR(PP))STEP2
510  IFPP/I<>INT(PP/I)THEN530
520  PR=0:RETURN
530  NEXTI:PR=1:RETURN
```

La routine « testpremier » cherche si le nombre transmis par la variable PP est premier et retourne le résultat de cette recherche dans la variable PR. PR est mise à 1 si PP est premier, et à 0 sinon.

Figure 9.22 : PP est-il Premier ?

```
# Souche pour entrée « groupée »
chargerimage   RETURN              # image actuelle conservée


# Souche pour choix de couples de nombres premiers
choix          RETURN              # dimensions actuelles conservées


# Souche pour sortie binaire du message
binaire        GOSUB effacecran
               FOR LL = 0 TO HL
                   FOR CC = 0 TO HC
                       IF D(LL, CC) = 1 THEN
                           PRINT "1"
                       else
                           PRINT "0";
                       NEXT CC
                   NEXT LL
               RETURN
```

```
540 RETURN
550 RETURN
560 GOSUB980:FORLL=0TOHL:FORCC=0TOHC:IFD(LL,CC)<>1THEN580
570 PRINT"1";:GOTO590
580 PRINT"0";
590 NEXTCC:NEXTLL:RETURN
```

La routine « chargerimage » est une souche qui fait immédiatement retour sans faire quoi que ce soit. Ce qui a pour effet de rappeler l'image précédente. Le but prévu de « chargerimage » est de charger en mémoire une chaîne de zéros et de uns antérieurement sauvegardés sur mémoire externe, ou bien frappés au clavier par le joueur.

La routine « choix » est une souche destinée à être utilisée avec « chargerimage ». Comme actuellement « choix » (ainsi que « chargerimage ») ne fait rien, c'est l'image précédente qui reviendra sur l'écran.

La routine « binaire » est une souche qui affiche sur l'écran la suite de uns et de zéros représentant l'image. La fonction prévue de « binaire » est la sauvegarde de l'image codée sur mémoire externe (disque, cassette) en vue du rechargement ultérieur par « chargerimage ».

***Figure 9.23 : Stockage sur Mémoire Externe** (Souches)*

— faire en sorte que la routine *divers* identifie une valeur de C2 égale à 1, comme signal d'appel de la routine *vie*

— réaliser la routine *vie* (fig. 9.44) et ses sous-programmes *voisins* (fig. 9.41, 9.42) et *viemort* (fig. 9.43).

Regardons ces différents points, étant donné que vous pouvez être tentés d'ajouter de nouvelles commandes de votre cru.

" L " a été définie comme « numéro 1 des diverses » par la ligne

DATA a(L), div, vie

de la figure 9.39. La signification de ces derniers symboles est donnée en figure 9.39. Le programme, qui interprète cet ordre DATA, est donné à la figure 9.38. Nous allons bientôt commenter ce programme.

Faire en sorte que *divers* comprenne qu'une valeur de C2 égale à 1, signifie un appel de la routine *vie*, a été réalisé en incorporant une comparaison explicite par rapport à 1. Si des commandes supplémentaires devaient être reconnues, il conviendrait de donner à *divers* une structure semblable à celle de *bougercurseur* (fig. 9.31). Les possibilités de l'instruction ON ... GOSUB conviennent tout à fait à ce problème d'aiguillage en fonction de la commande.

La réalisation de *vie* et de ses sous-programmes n'a pas présenté de difficultés particulières. Les sous-programmes *voisins* et *viemort* mettent en pratique les règles du Jeu de la Vie d'une façon tout à fait correcte, encore que l'utilisation qui est faite du tableau D pour compter les voisins mériterait une étude plus approfondie. Un tableau séparé pour compter les voisins aurait pu être utilisé, mais aurait pris nettement plus de place en mémoire.

```
# Mettre à zéro tableau des points

pointazero   FOR LX = 0 TO HL
             FOR CX = 0 O HC
                   D(LX, CX) = 0
                   NEXT CX
                NEXT LX
             RETURN

   600   FORLX=0TOHL:FORCX=0TOHC:D(LX,CX)=0:NEXTCX:NEXTLX:RETURN
```

La routine « pointazero » met à zéro le tableau D. Les éléments de D qui correspondent aux points de l'image seront mis à un.

Figure 9.24 : Mise à Zéro du Tableau des Points

La routine *vie* appelle *dessinerpoints* et *nouveaucurseur*. Elle utilise les dimensions d'image fixées par un appel préalable à *encadrer*. Pour le reste, *vie* se suffit à elle-même.

Finalement, examinons la relation existant entre certains caractères particuliers et les commandes. Cette relation est réalisée par la boucle *repeat ... until* de la routine *init* (fig. 9.38). Cette boucle permet d'utiliser des instructions DATA en nombre indéterminé. Ces instructions DATA sont présentées à la figure 9.39. Chacune contient un code ASCII, un code de commande primaire et un code de commande secondaire. Dans la boucle, le code ASCII est transformé en indice, qui permet d'accéder au tableau C. Quant aux codes de commande primaire et secondaire, ils sont condensés en un seul nombre qui est stocké dans l'élément de C indicé à partir du code ASCII.

Si vous désirez changer les codes de commande (par exemple, pour les adapter à un système n'ayant pas de clavier numérique séparé), il conviendrait de modifier les valeurs figurant dans les instructions DATA de la figure 9.39. De plus, si vous désirez utiliser des valeurs plus faibles ou plus grandes que celles définies par la fourchette CM (caractère maxi), CB (caractère de base), il vous faudra changer les valeurs de *petitcar* et *grandcar* de la figure 9.38. Ces valeurs sont utilisées pour calculer la dimension du tableau C.

Améliorations et Additions

Voici quelques-unes des nombreuses améliorations et additions qu'il est possible d'apporter à la Vie Extra-Terrestre :

- Réaliser le stockage externe des images binaires. Les routines *chargerimage, choix* et *binaire* (fig. 9.23) sont les seules à modifier.

```
# Calcul du coin supérieur gauche pour centrage du cadre
centrer   CZ = INT((SC - HC - 3)/2) + 1    # concerne le cadre
          LZ = INT((SL - HL - 3)/2) + 1
          RETURN

610  CZ=INT((SC-HC-3)/2)+1:LZ=INT((SL-HL-3)/2)+1:RETURN
```

La routine « centre » détermine la position du coin supérieur gauche du cadre de façon que l'image soit centrée sur l'écran.

Figure 9.25 : Centrage de l'Image

```
# Tracer le cadre entourant la zone image

dessinercadre  L = LZ - 1 : C = CZ : GOSUB curseur    # en haut à gauche
               FOR CC = 0 TO HC                        # partie supérieure
                    PRINT TH$;
                    NEXT CC
               PRINT "vers le bas"
               FOR LL = 0 TO HL                        # partie droite
                    PRINT RV$; "en arrière"; "vers le bas"
                    NEXT LL
               L = LZ : C = CZ - 1 : GOSUB curseur    # en haut à gauche
               FOR LL = 0 TO HL                        # partie gauche
                    PRINT LV$; "en arrière"; "vers le bas";
               NEXT LL
               PRINT "vers la droite";
               FOR CC = 0 TO HC                        # partie inférieure
                    PRINT BH$;
                    NEXT CC
               RETURN
```

```
620* L=LZ-1:C=CZ:GOSUB940:FORCC=0TOHC:PRINTTH$;:NEXTCC:
     PRINTCHR$(17);
630* FORLL=0TOHL:PRINTRV$;CHR$(157);CHR$(17);:NEXTLL
640* L=LZ:C=CZ-1:GOSUB940:FORLL=0TOHL:PRINTLV$;CHR$(157);CHR$(17);:
     NEXTLL:PRINTCHR$(29);
650  FORCC=0TOHC:PRINTBH$;:NEXTCC:RETURN
```

La routine « dessinercadre » trace le cadre entourant la zone image. Les caractères de déplacement du curseur sont utilisés sur le Pet ou le TRS-80, mais des routines très voisines sont utilisables sur Apple.

Les commandes de déplacement de curseur de l'Apple sont la solution du problème sur ces derniers systèmes. Les instructions BASIC réelles sont sur Apple :

```
620  L=LZ-1:C=CZ:GOSUB940: FOR CC=0 TO HC: PRINT TH$;: NEXT CC
630  FOR LL=0 TO HL:L=LZ+LL: C=CZ-1:GOSUB940: PRINT LV$;
640  C=CZ+HC+1:GOSUB940: PRINT RV$;: NEXT LL
650  L=LZ+HL+1:C=CZ:GOSUB940: FOR CC=0 TO HC: PRINT BH$;: NEXT CC:
     RETURN
```

* Sur TRS-80, les caractères curseur arrière, vers le bas et à droite, sont représentés par CHR$(24), CHR$(26) et CHR$(25).

Figure 9.26 : Le Tracé du Cadre

```
# Dessiner une image d'après tableau D
point/blanc  FOR LL = 0 TO HL
                 L = LL + LZ : C = CZ : GOSUB curseur   # mettre le curseur en
                                                          début de ligne
                 FOR CC = 0 TO HC
                     IF D(LL, CC) < > 0 THEN
                         PRINT DT$;                      # afficher un « point »
                     else
                         PRINT " ";                      # afficher un blanc
                     NEXT CC
                 NEXT LL

660  FORLL=0TOHL:L=LL+LZ:C=CZ:GOSUB940
670  FORCC=0TOHC:IFD(LL,CC)<>0THENPRINTDT$;:GOTO690
680  PRINT" ";
690  NEXTCC:NEXTLL:RETURN
```

La routine « point/blanc » redessine l'image, mettant à chaque position soit un point, soit un blanc et de ce fait effaçant les anciens points. Cette routine sert d'une part à dessiner chaque nouvelle génération dans le Jeu de la Vie, et d'autre part à redessiner une ancienne image qu'on vient de rappeler.

Figure 9.27 : Dessin de l'Image à Partir du Tableau D

```
# Saisir et décoder une commande de dessin mono-caractère
cmd/dessin  GOSUB entreecurseur                # faire clignoter curseur image,
                                                 attendre entrée
            CX = ASC(X$) - CB
            IF NOT 0 < = CX < = MX THEN        # indice tableau C
                CX = DX                        # valeur par défaut si hors limites
            C1 = INT(C(CX)/100)                # commande primaire
            C2 = C(CX) - 100 * C1              # commande secondaire
            RETURN

700  GOSUB1080:CX=ASC(X$)-CB:IFCX>MXORCX<0THENCX=DX
710  C1=INT(C(CX)/100):C2=C(CX)-100*C1:RETURN
```

La routine « cmd/dessin » transforme une commande mono-caractère entrée par le joueur en une commande primaire C1 et une commande secondaire C2. La commande primaire est analysée dans la routine principale. Dans la plupart des cas, la commande secondaire est interprétée dans la routine « bougercurseur » (fig. 9.31). Les commandes secondaires correspondant à une commande primaire « diverse » sont interprétées dans la routine « divers » (fig. 9.30).

Figure 9.28 : Le Décodage des Commandes

Commandes de dessin primaires (indicées par C1)

```
deplacement   GOSUB bougercurseur    # simple déplacement du curseur
              RETURN

creermarque   GOSUB ecriremarque     # mettre un point, puis déplacer le curseur
              GOSUB bougercurseur
              RETURN

tuermarque    GOSUB effacermarque    # effacer un point, puis déplacer le curseur
              GOSUB bougercurseur
              RETURN

cmd/div       GOSUB divers           # travaux divers
              RETURN
```

```
720 GOSUB780:RETURN
730 GOSUB800:GOSUB780:RETURN
740 GOSUB810:GOSUB780:RETURN
750 GOSUB760:RETURN
```

Voici les routines appelées par le programme principal (fig. 9.15) pour le traitement des commandes primaires. Chaque routine exécute d'abord une action (qui peut consister à ne rien faire), puis appelle une autre routine qui exécute la commande secondaire.

Figure 9.29 : Traitement des Commandes Primaires

```
# Routine des commandes diverses

divers  IF C2 = 1 THEN                # C2 = 1 est le code pour le Jeu de la Vie
            GOSUB vie
        RETURN

  760  IFC2=1THENGOSUB1450
  770  RETURN
```

La routine « divers » est l'endroit du programme où l'on peut ajouter de nouvelles commandes Par exemple, si l'on désirait avoir une commande permettant d'effacer l'image entière, une instruction DATA adéquate pourrait être ajoutée à la routine d'initialisation (fig. 9.39) indiquant une valeur de 4 (divers) pour la commande primaire et une valeur de 2 pour la commande secondaire. Une modification de « divers » permettrait de rendre significative un C2 égal à 2 et d'appeler une routine qui consisterait à faire appel à « pointazero » puis à « point/blanc ». Alors à chaque fois que le joueur entrerait le caractère précisé dans l'ordre DATA, toute l'image (mais pas le cadre) serait effacée.

Figure 9.30 : Actions Diverses

```
# Aiguillage vers l'une des routines de déplacement du curseur

bougercurseur  IF 1 < = C2 < = 4 THEN    # vérifier C2 dans limites
                   ON C2 GOSUB
                       curseurgauche     # déplacer curseur à gauche
                       curseurdroite,    # déplacer curseur à droite
                       curseurhaut,      # déplacer curseur vers le haut
                       curseurbas        # déplacer curseur vers le bas
               RETURN

  780  IFC2<1ORC2>4THENRETURN
  790  ONC2GOSUB860,880,900,920:RETURN
```

La routine « bougercurseur » interprète la commande secondaire liée aux commandes primaires « déplacement », « créermarque » et « tuermarque ». La commande secondaire dit où le curseur doit aller après que la commande primaire ait été exécutée.

Figure 9.31 : Indiquer au Curseur où Aller

Routines pour afficher ou effacer des points

ecriremarque	GOSUB creerpoint	# mettre un nouveau point dans l'image
	D(LL, CC) = 1	# en LL, CC et mémoriser
	RETURN	# ce point dans le tableau D
effacermarque	GOSUB tuerpoint	# supprimer le point situé en LL, CC
	D(LL, CC) = 0	# et mettre l'élément correspondant de D à zéro
	RETURN	
otercurseur	IF D(LL, CC) = 1 THEN	# si le curseur masquait un point,
	GOSUB creerpoint	# remettre le point
	else	
	GOSUB tuerpoint	# sinon, ôter toutes les marques
	RETURN	
nouveaucurseur	IF D(LL, CC) = 1 THEN	# si le curseur doit masquer un point,
	CS$ = OD$	# afficher le curseur type « avec point »
	else	
	CS$ = ND$	# sinon, afficher curseur type « sans point »
	RETURN	

```
800  GOSUB1170:D(LL,CC)=1:RETURN
810  GOSUB1180:D(LL,CC)=0:RETURN
820  IFD(LL,CC)=1THENGOSUB1170:RETURN
830  GOSUB1180:RETURN
840  CS$=ND$:IFD(LL,CC)=1THENCS$=OD$
850  RETURN
```

Ces routines affichent ou effacent des points. En plus des points ordinaires qui constituent l'image, on utilise deux caractères pour le curseur d'image. Le premier sert lorsque le curseur se trouve à un emplacement blanc, le deuxième lorsqu'il masque un point.

Figure 9.32 : Affichage des Points et des Curseurs

Routines de déplacement du curseur d'image

```
curseurgauche   IF CC > 0 THEN
                    CC = CC - 1
                else
                    CC = HC
                RETURN

curseurdroite   IF CC < HC THEN
                    CC = CC + 1
                else
                    CC = 0
                RETURN

curseurhaut     IF LL > 0 THEN
                    LL = LL - 1
                else
                    LL = HL
                RETURN

curseurbas      IF LL < HL THEN
                    LL = LL + 1
                else
                    LL = 0
                RETURN
```

```
860  IFCC>0THENCC=CC-1:RETURN
870  CC=HC:RETURN
880  IFCC<HCTHENCC=CC+1:RETURN
890  CC=0:RETURN
900  IFLL>0THENLL=LL-1:RETURN
910  LL=HL:RETURN
920  IFLL<HLTHENLL=LL+1:RETURN
930  LL=0:RETURN
```

Ces routines calculent la nouvelle position (LL, CC) du curseur d'image, en réponse à une commande de déplacement du curseur. Les valeurs de la position du curseur sont contenues dans les intervalles 0< =LL< =HL et 0< =CC< =HC. Les valeurs « enroulent », sautant de HC ou de HL à zéro quand elles dépassent la limite supérieure, et sautant de 0 à HC ou HL dès qu'elles ont tendance à devenir négatives.

Figure 9.33 : Les Déplacements du Curseur d'Image

```
# Mettre le vrai curseur en colonne C, ligne L

curseur PRINT " en haut à gauche ";      # mettre dans le coin supérieur gauche
        C = C mod SC                     # garder position dans limites de l'écran
        L = L mod SL
        IF C > 0 THEN                    # s'il n'est pas sur la colonne de gauche
            FOR ZZ = 1 TO C
            PRINT " à droite ";          # PRINT caractères " curseur à droite "
                NEXT ZZ
        IF L > 0 THEN                    # si ce n'est pas la ligne du haut
            FOR ZZ = 1 TO L
                PRINT " en bas ";        # PRINT caractères " curseur en bas "
                NEXT ZZ
        RETURN
```

```
940* PRINTCHR$(19);:C=C-SC*INT(C/SC):L=L-SL*INT(L/SL)
950* IFC>0THENFORZZ=1TOC:PRINTCHR$(29);:NEXTZZ
960* IFL>0THENFORZZ=1TOL:PRINTCHR$(17);:NEXTZZ
970  RETURN
```

La routine « curseur » est semblable à d'autres routines figurant dans d'autres programmes de ce livre. Ici le curseur vidéo est placé à la colonne C et à la ligne L ; partout ailleurs dans le présent ouvrage, les arguments sont CC et LL. Dans la Vie Extra-Terrestre, on ne peut pas se servir de CC et LL comme arguments de « curseur » parce que ces variables sont déjà utilisées pour le curseur d'image. Ceci met en évidence le problème fondamental de l'utilisation de sous-programmes en BASIC : il n'existe pas d'arguments fictifs. Les arguments sont passés aux sous-programmes dans des variables globales.

Les instructions en BASIC réel de la version Apple sont :

```
940  C=C-SC*INT(C/SC):L=L-SL*INT(L/SL)
950  HTAB C+1
960  LTAB L+1
970  RETURN
```

* Sur TRS-80, les caractères « en haut à gauche », « à droite » et en bas sont CHR$(28), CHR$(25) et CHR$(26).

Figure 9.34 : Position du Curseur Vidéo

```
# Effacer l'écran

effacecran      PRINT "clr";
                RETURN

# Entrée mono-caractère

uncar           repeat
                    GET X$
                    until (X$ <> "")

# Entrée chaîne

entreechaine    XX$ = ""
                repeat {
                    GOSUB uncar
                    IF X$ <> "effacement" THEN {
                        PRINT X$;
                        IF XS <> "return" THEN
                            XX$ = XX$ + X$
                        else break
                        }
                    else IF LEN(XX$) <> 0 THEN {        # s'il reste au moins
                                                        un caractère
                        PRINT                           # alors effacer le
                          "caractère d'effacement"      plus à droite
                        XX$ = LRFT$(XX$, LEN(XX$) - 1)
                        }
                    }
                RETURN

 980 PRINTCHR$(147);:RETURN
 990 GETX$:IFX$=""THEN990
1000 RETURN
1010 XX$=""
1020 GOSUB990:IFX$=CHR$(20)THEN1050
1030 PRINTX$;:IFX$=CHR$(13)THEN1070
1040 XX$=XX$+X$:GOTO1020
1050 IFLEN(XX$)=0THEN1020
1060 PRINTX$;:XX$=LEFT$(XX$,LEN(XX$)-1):GOTO1020
1070 RETURN
```

Ces trois routines sont identiques à celles qui figurent dans d'autres programmes de ce livre. (Voir figure 4.10 pour leurs versions Apple et TRS-80.)

Figure 9.35 : Routines Utilitaires déjà Rencontrées

```
# Entrer un caractère pendant que le curseur est affiché — Version pour le Pet

entréecurseur   GOSUB nouveaucurseur          # choisir le caractère qui sera le
                                              curseur
                CF = 1                        # boucle pour faire clignoter
                repeat {
                    IF CF = 1 THEN
                        GOSUB curseurallumé   # curseur en circuit
                    else
                        GOSUB curseureteint   # curseur hors circuit
                    CF = - CF : TZ = TI       # changer aiguillage; noter
                                              heure
                    repeat {                  # attendre entrée
                        GET X$
                        IF X$ <> " " THEN
                          { GOSUB             # ôter le curseur
                          otercurseur : RETURN }
                    } until (TI - TZ >= FQ)   # il est temps de faire clignoter
                }

curseurallumé   C = CZ + CC : L = LZ + LL : GOSUB curseur
                PRINT CS$; : RETURN

curseureteint   C = CZ + CC : L = LZ + LL : GOSUB curseur
                PRINT " "; : RETURN

1080  GOSUB840:CF=1
1090  IFCF=1THENGOSUB1150:GOTO1110
1100  GOSUB1160
1110  CF=-CF:TZ=TI
1120  GETX$:IFX$<>""THENGOSUB820:RETURN
1130  IFTI-TZ<FQTHEN1120
1140  GOTO1090
1150  C=CZ+CC:L=LZ+LL:GOSUB940:PRINTCS$;:RETURN
1160  C=CZ+CC:L=LZ+LL:GOSUB940:PRINT" ";:RETURN
```

La routine « entréecurseur » et ses deux sous-programmes « curseurallumé » et « curseuréteint » réalise le clignotement du curseur d'image, en utilisant pour la mesure de la durée les possibilités offertes par la variable TI du Pet. Sur TRS-80 et sur Apple, les solutions sont différentes. La fonction GET de l'Apple fait clignoter un curseur (contrairement au GET du Pet). Sur TRS-80 il faut utiliser une « boucle de temporisation ». La version TRS-80 se trouve en figure 9.36*a*. Les instructions en BASIC Réel pour l'Apple sont :

```
1080  C=CZ+CC:L=LZ+LL:GOSUB940:GOSUB990:RETURN
```

Figure 9.36 : Faire Clignoter le Curseur en Attendant la Commande

```
1080 GOSUB840:CF=1
1090 IFCF=1THENGOSUB1150:GOTO1110
1100 GOSUB1160
1110 CF=-CF
1120 FORQQ=1TOCJ*FQ:X$=INKEY$:IFX$<>""THENGOSUB820:RETURN
1130 NEXTQQ
1140 GOTO1090
1150 C=CZ+CC:L=LZ+LL:GOSUB940:PRINTCS$;:RETURN
1160 C=CZ+CC:L=LZ+LL:GOSUB940:PRINT"";:RETURN
```

Figure 9.36a : BASIC Réel TRS-80 pour « entréecurseur »

```
# Mettre un point en LL, CC

creerpoint  L = LZ + LL : C = CC + CZ : GOSUB curseur
            PRINT DT$;
            RETURN


# Effacer le point en LL, CC

tuerpoint   L = LZ + LL : C = CC + CZ : GOSUB curseur
            PRINT "";
            RETURN

1170 C=CZ+CC:L=LZ+LL:GOSUB940:PRINTDT$;:RETURN
1180 C=CZ+CC:L=LZ+LL:GOSUB940:PRINT"";:RETURN
```

Les routines « créerpoint » et « tuerpoint » affichent un point ou l'effacent à la position LL, CC.

Figure 9.37 : Création et Mise à Mort des Points

```
# Initialisation de la Vie Extra-Terrestre
init  READ SL, SC : DATA lignes, colonnes                # taille écran
      READ BH$, TH$, LV$, RV$                            # éléments de lignes pour
      DATA " bas ", " haut ", " gauche ", " droite "     # cadre de l'image
      READ OD$, NDS : DATA "avec point", "sans point"    # caractères du curseur
                                                         d'image
      READ FQ : DATA periode                             # vitesse de clignotement
      READ DT$ : DATA " point "                          # caractère point image
      READ MC, ML, RL                                    # taille maxi idéale pour
                                                         image et
      DATA maxcols, maxlignes, limite mémoire            # limite imposée par la
                                                         mémoire
      MC = min(MC, SC - 2, RL)                           # nombre maxi colonnes
                                                         image
      ML = min(ML, SL - 2, RL)                           # nombre maxi lignes
                                                         image
      DIM D(ML, MC)                                      # tableau des points de
                                                         l'image
      READ CB, CM : DATA petitcar, grandcar              # bornes des carac. de
                                                         commande ASCII
      MX = CM - CB : IF MX < 1 THEN STOP                 # dimension du tableau
                                                         des commandes
      DIM C(MX)
      FOR CY = 0 TO MX
           C(CY) = div. * 100 + rienfaire                # remplir le tableau des
                                                         commandes de codes rien-
                                                         faire

           NEXT CY
      repeat {                                           # charger le tableau des
                                                         commandes
           READ CH                                       # code ASCII du carac-
                                                         tère
           IF CH <> - 1 THEN {
                CY = CH - CB                             # indice pour ce caractère
                IF NOT 0 < = CY < = MX THEN STOP
                READ C1, C2                              # codes de commande
                C(CY) = 100 * C1 + C2                    # primaire et secondaire
                }
           } until (CH = - 1)                            #   - 1 indique la fin
      # Instructions DATA pour le tableau des commandes (voir figure 9.39)
      # Format : DATA code ASCII, commande primaire, commande secondaire
      READ CH : DATA defautcar                           # pour commandes hors
                                                         limites
      DX = CH - CB                                       # indice par défaut pour
                                                         entrer dans le tableau C
      RETURN
```

(Les instructions en BASIC Réel sont données en figure 9.40)

Figure 9.38 : Initialisation de la Vie Extra-Terrestre

```
#  Instructions DATA pour le tableau des commandes (extrait de la routine d'initialisation)
#  Format : a(caractère), commande primaire, commande secondaire
#  a(caractère) signifie code ASCII du caractère correspondant
#  — 1 à la place d'un code ASCII signale la fin de la liste

#  Les commandes primaires peuvent prendre les valeurs suivantes :
#     0     exit            (l'image est écrite sous forme de zéros et de uns)
#     1     déplacement     (déplacer le curseur d'image)
#     2     creerpoint      (faire un point puis déplacer le curseur)
#     3     tuerpoint       (effacer un point, puis déplacer le curseur)
#     4     div             (autres commandes)

#  Commandes secondaires (pour déplacement, créerpoint, tuerpoint) :
#     1     gauche          (le curseur d'image va sur la gauche)
#     2     droite          (le curseur d'image va sur la droite)
#     3     haut            (le curseur d'image va vers le haut)
#     4     bas             (le curseur d'image va vers le bas)

#  Commandes secondaires (pour div) :
#     0     rienfaire       (ne rien faire)
#     1     vie             (transformer l'image d'après les règles du Jeu de la Vie)

DATA  a(4),         deplacement,  gauche,  a(6),         deplacement,  droite
      a(8),         deplacement,  haut,    a(2),         deplacement,  bas
DATA  a(shift 4),   creerpoint,   gauche,  a(shift 6),   creerpoint,   droite
      a(shift 8),   creerpoint,   haut,    a(shift 2),   creerpoint,   bas
DATA  a(0),         tuerpoint,    gauche,  a(5),         tuerpoint,    droite
      a(7),         tuerpoint,    haut,    a(1),         tuerpoint,    bas
DATA  a(L),         div,          vie .
DATA  a(E),         exit,         0,       — 1
```

Les instructions DATA précisent les commandes mono-caractères comprises par le programme. Le tableau C comporte un élément pour chacun des codes ASCII dans les limites correspondant aux valeurs de CB et CM. La commande correspondant à CB est stockée en C(0). Chaque commande se présente sous forme d'un nombre de 2 chiffres en base 100. C'est-à-dire que si C1 et C2 sont respectivement les commandes primaire et secondaire à stocker en C(CY), alors la valeur donnée à C(CY) sera 100 * C1 + C2.

La valeur — 1 finale qui termine la liste permet d'allonger ou de raccourcir celle-ci facilement.

Figure 9.39 : Initialisation des Valeurs des Commandes

```
1190*     READSL,SC:DATA24,40
1210**    READBH$,TH$,LV$,RV$:DATA"—","_","|","|"
1220**    READOD$,ND$:DATA"◙","□"
1230**    READFQ:DATA18
1240**    READDT$:DATA"•»
1250***   READMC,ML,RL:DATA37,37,16
1260      IFMC>SC-2THENMC=SC-2
1270      IFMC>RLTHENMC=RL
1280      IFML>SL-2THENML=SL-2
1290      IFML>RLTHENML=RL
1300      DIMD(ML,MC)
1310**    READCB,CM:DATA48,185
1320      MX=CM-CB:IFMX<1THENSTOP
1330      DIMC(MX)
1340      FORCY=0TOMX:C(CY)=400:NEXTCY
1350      READCH:IFCH=-1THEN1380
1360      CY=CH-CB:IFCY<0ORCY>MXTHENSTOP
1370      READC1,C2:C(CY)=100*C1+C2:GOTO1350
1380      DATA52,1,1,54,1,2,56,1,3,50,1,4
1390****  DATA180,2,1,182,2,22,2,2,184,2,3,178,2,4
1400      DATA48,3,1,53,3,2,55,3,3,49,3,4
1410      DATA76,4,1
1420      DATA69,0,0,-1
1430      READCH:DATA52
1440      DX=CH-CB:RETURN
```

Voici les instructions en BASIC réel correspondant à la routine « init » présentée aux figures 9.38 et 9.39. La routine « init » précise les valeurs de nombreuses variables qui ont pris la place de constantes tout au long du programme. L'utilisation de variables facilite les changements qui sont nécessaires chaque fois que l'on veut adapter le programme à un autre type d'ordinateur individuel.

La technique utilisée pour l'initialisation du tableau C mérite de retenir l'attention.

---

* Sur TRS-80, les dimensions de l'écran sont 16 et 64.

** Les versions Apple et TRS-80 de ces lignes sont les suivantes :

```
1210  READ BH$,TH$,LV$,RV$:DATA "-","-","!","!"
1220  et 1230 non utilisées sur Apple.
1220  READOD$,ND$:DATA"#","*"              } pour le TRS-80
1230  READFQ:DATA18:READCJ:DATA.25         }
1240  READ DT$:DATA "0"
1310  READ CB,CM:DATA 0, 127
```

*** La valeur de 16 pour RL permet au programme de tourner sur un Pet de 8K. Pour des systèmes plus importants on peut utiliser 100, ce qui revient à dire qu'il n'y a pas de limite pratique concernant la taille du tableau, puisque tout ce qui tient sur l'écran trouvera place en mémoire.

**** Sur Apple et TRS-80, 180, 182, 184 et 178 sont remplacés par 36, 38, 40 et 34.

Figure 9.40 : Initialisation de la Vie Extra-Terrestre *(BASIC Réel)*

```
# Compter les voisins

voisins  IF D(0, 0) est impair THEN
                ajouter 2 à D(0, 1), D(1, 0), D(1, 1)

         IF D(0, HC) est impair THEN
                ajouter 2 à D(0, HC − 1), D(1, HC), D(1, HC − 1)

         IF D(HL, 0) est impair THEN
                ajouter 2 à D(HL, 1), D(HL − 1, 0), D(HL − 1, 1)

         IF D(HL, HC) est impair THEN
                ajouter 2 à D(HL, HC − 1), D(HL − 1, HC), D(HL − 1, HC − 1)

         IF HC > 1 THEN
                FOR CC = 1 TO HC − 1
                       IF D(0, CC) est impair THEN
                              ajouter 2 à D(0, CC − 1), D(0, CC + 1),
                              D(1, CC − 1), D(1, CC), D(1, CC + 1)
                       IF D(HL, CC) est impair THEN
                              ajouter 2 à D(HL, CC − 1), D(HL, CC + 1),
                              D(HL − 1, CC − 1), D(HL − 1, CC), D(HL − 1, CC + 1)
                       NEXT CC

         IF HL > 1 THEN
                FOR LL = 1 TO HL − 1
                       IF D(LL, 0) est impair THEN
                              ajouter 2 à D(LL − 1, 0), D(LL + 1, 0),
                              D(LL − 1, 1), D(LL, 1), D(LL + 1, 1)
                       IF D(LL, HC) est impair THEN
                              ajouter 2 à D(LL − 1, HC), D(LL + 1, HC),
                              D(LL − 1, HC − 1), D(LL, HC − 1), D(LL + 1, HC − 1)
                       IF HC > 1 THEN
                              FOR CC = 1 TO HC − 1
                                     IF D(LL, CC) est impair THEN
                                            ajouter 2 à D(LL, CC − 1), D(LL, CC + 1),
                                            D(LL−1, CC−1), D(LL−1, CC), D(LL−1, CC+1),
                                            D(LL+1, CC−1), D(LL+1, CC), D(LL+1, CC+1)
                                     NEXT CC
                       NEXT LL

         RETURN
```

Figure 9.41 : Le Recensement des Voisins

```
1460  D=D(0,0)/2:IFD=INT(D)THEN1480
1470  D(0,1)=D(0,1)+2:D(1,0)=D(1,0)+2:D(1,1)=D(1,1)+2
1480  D=D(0,HC)/2:IFD=INT(D)THEN1500
1490  D(0,HC-1)=D(0,HC-1)+2:D(1,HC)=D(1,HC)+2:D(1,HC-1)=D(1,HC-1)+2
1500  D=D(HL,0)/2:IFD=INT(D)THEN1520
1510  D(HL,1)=D(HL,1)+2:D(HL-1,0)=D(HL-1,0)+2:D(HL-1,1)=D(HL-1,1)+2
1520  D=D(HL,HC)/2:IFD=INT(D)THEN1550
1530  D(HL,HC-1)=D(HL,HC-1)+2:D(HL-1,HC)=D(HL-1,HC)+2
1540  D(HL-1,HC-1)=D(HL-1,HC-1)+2
1545  IFHC<=1THEN1615
1550  FORCC=1TOHC-1:D=D(0,CC)/2:IFD=INT(D)THEN1580
1560  D(0,CC-1)=D(0,CC-1)+2:D(0,CC+1)=D(0,CC+1)+2:D(1,CC-1)
      =D(1,CC-1)+2
1570  D(1,CC)=D(1,CC)+2:D(1,CC+1)=D(1,CC+1)+2
1580  D=D(HL,CC)/2:IFD=INT(D)THEN1610
1590  D(HL,CC-1)=D(HL,CC-1)+2:D(HL,CC+1)=D(HL,CC+1)+2:
      D(HL-1,CC)=D(HL-1,CC)+2
1600  D(HL-1,CC-1)=D(HL-1,CC-1)+2:D(HL-1,CC+1)=D(HL-1,CC+1)+2
1610  NEXTCC
1615  IFHL<=1THENRETURN
1620  FORLL=1TOHL-1:D=D(LL,0)/2:IFD=INT(D)THEN1650
1630  D(LL-1,0)=D(LL-1,0)+2:D(LL+1,0)=D(LL+1,0)+2:D(LL-1,1)=
      D(LL-1,1)+2
1640  D(LL,1)=D(LL,1)+2:D(LL+1,1)=D(LL+1,1)+2
1650  D=D(LL,HC)/2:IFD=INT(D)THEN 1675
1660  D(LL-1,HC)=D(LL-1,HC)+2:D(LL+1,HC)=D(LL+1,HC)+2:
      D(LL,HC-1)=D(LL,HC-1)+2
1670  D(LL-1,HC-1)=D(LL-1,HC-1)+2:D(LL+1,HC-1)=D(LL+1,HC-1)+2
1675  IFHC<=1THENNEXTLL:RETURN
1680  FORCC=1TOHC-1:D=D(LL,CC)/2:IFD=INT(D)THEN1730
1690  D(LL,CC-1)=D(LL,CC-1)+2:D(LL,CC+1)=D(LL,CC+1)+2
1700  D(LL-1,CC-1)=D(LL-1,CC-1)+2:D(LL-1,CC+1)=D(LL-1,CC+1)+2
1710  D(LL+1,CC-1)=D(LL+1,CC-1)+2:D(LL+1,CC+1)=D(LL+1,CC+1)+2
1720  D(LL-1,CC)=D(LL-1,CC)+2:D(LL+1,CC)=D(LL+1,CC)+2
1730  NEXTCC:NEXTLL:RETURN
```

La routine « voisins » enregistre le nombre des voisins de chaque point en ajoutant 2 fois ce nombre à la valeur (0 ou 1) de l'élément correspondant du tableau D. La routine explore la totalité de l'image et quand elle trouve un point, le nombre des voisins de chaque position avoisinante est incrémenté. La routine est compliquée par le fait que les positions qui sont près des bords n'ont pas huit positions avoisinantes. Cette routine pourrait être grandement simplifiée par l'utilisation d'une « bordure » factice dans le tableau D car chaque point aurait alors 8 positions avoisinantes. Mais les positions faisant partie de cette « bordure » ne contiendraient jamais de points.

Figure 9.42 : Le Recensement des Voisins *(BASIC Réel)*

```
# Prendre des décisions de vie ou de mort

viemort  FOR LL = 0 TO HL
             FOR CC = 0 TO HC
                 NB = INT(D(LL, CC)/2)     # 2 * nbre de voisins est stocké en D
                 PT = D(LL, CC) - 2 * NB   # la valeur de la position est celle du
                                           « bit le moins significatif »
                 IF case
                     NB < 2 OR NB > 3 THEN
                         D(LL, CC) = 0     # pas de point
                     NB = 3 THEN
                         D(LL, CC) = 1     # un point
                     NB = 2 THEN
                         D(LL, CC) = PT    # maintenir le statu quo
                 NEXT CC
             NEXT LL
         RETURN
```

```
1740  FORLL=0TOHL:FORCC=0TOHC:D=D(LL,CC):NB=INT(D/2):PT=D-2*NB
1750  IFNB<2ORNB>3THEND(LL,CC)=0:GOTO1780
1760  IFNB=3THEND(LL,CC)=1:GOTO1780
1770  D(LL,CC)=PT
1780  NEXTCC:NEXTLL:RETURN
```

La routine « viemort » décide que telle position doit ou ne doit pas avoir de point. L'algorithme utilisé est le suivant :

— Si la position a moins de 2, ou plus de 3 voisins, il ne doit pas y avoir de point.
— Si la position a 2 voisins, la nouvelle génération n'aura de point que s'il y en existait déjà un.
— Si la position a 3 voisins, il doit y avoir un point.

Au fur et à mesure que le tableau D est calculé à partir de l'ancien, le décompte des voisins disparaît et les valeurs de chaque élément du tableau redeviennent soit un soit zéro.

Figure 9.43 : Où Doit-il y Avoir des Points ?

- Eliminer l'appel à *point/blanc* dans la routine *encadrer* (fig. 9.16) quand un nouveau cadre vient d'être spécifié. Cette modification accélérera l'initialisation du programme.
- Ajouter de nouvelles commandes de dessin :
  — Des commandes qui déplacent d'abord le curseur, puis exécutent l'action.
  — Des commandes qui déplacent le curseur en « diagonale ».
  — Une commande pour « défaire » ce qu'a fait la dernière commande.
- Modifier la routine *voisins* pour pratiquer « l'enroulement ». C'est-à-dire considérer que la position (LL, HC) est voisine de (LL, 0) ou encore que (0, CC) jouxte (HL, CC). De cette façon, chaque point a huit positions avoisinantes. Le jeu devient le Jeu de la Vie sur un « tore ». (« Tore » est le nom mathématique pour une surface ayant la forme d'une chambre à air.) Voyez-vous pourquoi on peut parler de tore ?
- Accélérer les transformations du Jeu de la Vie.

En résumé

Le jeu de la Rencontre des Extra-Terrestres permet au joueur de concevoir des messages sous forme d'images, à l'aide de matrices

```
# Transformer le dessin selon les règles du Jeu de la Vie
vie  GOSUB voisins : GOSUB viemort : GOSUB point/blanc   # transformer et redessiner
     CC = 0 : LL = 0 : GOSUB nouveaucurseur : RETURN      # dans le coin en haut à gauche

  1450  GOSUB1460:GOSUB1740:GOSUB660:CC=0:LL=0:GOSUB840:RETURN
```

La routine « vie » est appelée à partir de « divers » pour réaliser une transformation de l'image d'après les règles du Jeu de la Vie. On appelle d'abord « voisins » pour compter les voisins qu'à chaque position. (Un voisin est un point occupant l'une des huit positions jouxtant une position donnée.) Puis « viemort » est appelée pour transformer le contenu du tableau D (c'est-à-dire les points de l'image) conformément aux règles du Jeu de la Vie, lesquelles dépendent du nombre de voisins qu'a chaque position. Finalement « point/blanc » est chargée de redessiner l'image et la place du curseur d'image est fixée comme étant en haut et à gauche de l'écran par « nouveaucurseur ».

Figure 9.44 : Les Règles de la Vie sont Appliquées à l'Image

de points. Ces messages sont destinés à être transmis aux extra-terrestres, sous forme de suite de bits. Le nombre de bits constituant cette suite doit être le produit de deux nombres entiers. Ce sont les messages de ce type qui ont le plus de chances d'être correctement interprétés par un destinataire qui doit en deviner le format.

Le jeu bien connu sous le nom de Jeu de la Vie a été greffé sur la Rencontre des Extra-Terrestres, de façon à ce que l'image réalisée puisse subir des transformations conformes aux règles du Jeu de la Vie.

Le programme, qui réalise le jeu hybride de la Vie Extra-Terrestre, a deux aspects importants : la zone image et les commandes. Les éléments de la zone image sont le cadre, les points du dessin et le curseur d'image. Les routines *encadrer*, *centrer* et *dessinercadre* permettent le tracé du cadre. Les points de l'image sont le reflet du tableau D, et sont affichés par la routine *point/blanc*. Le curseur d'image est contrôlé par la routine *entréecurseur*, qui précise le caractère employé et la position du curseur, réalise le clignotement et rétablit l'affichage convenable lorsque le curseur est supprimé.

Les commandes mono-caractères reconnues par le programme de la Vie Extra-Terrestre, sont précisées dans la routine *init*. Des caractères déterminés sont associés à des commandes précises par l'intermédiaire du tableau C, dont les indices sont tirés des codes ASCII des caractères et dont les différents éléments contiennent des nombres qui codent simultanément la commande primaire et la commande secondaire. La commande primaire permet de choisir entre « exit », « dessin » ou « divers ». La commande secondaire permet de détailler chacune des deux dernières catégories. Le Jeu de la Vie est codé sous l'étiquette « commandes diverses ».

De nombreuses améliorations et additions sont possibles, et tout d'abord le stockage externe des images numériques.

APPENDICE **A**

Le Jeu de Caractères ASCII

La table ci-contre montre le jeu « standard » des caractères ASCII. Les codes numériques (base 10) vont en croissant de 0 à 127 et le caractère correspondant est indiqué pour chacun d'eux.

Tous les systèmes BASIC utilisent ce code avec de légères variantes. Le programme suivant aidera à déterminer les valeurs décimales des codes affectés à des caractères spéciaux dans votre système :

```
1 GOSUB 2 : PRINT ASC(X$) ; " ";: GOTO 1
2 (Routine d'entrée mono-caractère présentée pour
  l'Apple, le Pet et le TRS-80 en figure 2.11. Les lignes 720
  et 730 de la figure 2.11 sont ici devenues 2 et 3.)
```

Lorsque vous aurez entré ce programme et tapé RUN, vous pourrez appuyer sur n'importe quelle touche pour voir immédiatement la valeur numérique du code correspondant.

En plus des codes allant de 0 à 127, le Pet et le TRS-80 utilisent les codes 128 à 255 à des fins particulières. Sur TRS-80, les codes 128 à 191 représentent des « blocs graphiques » et les codes 192 à 255, des chaînes de blancs*. Sur Pet, les codes 128 à 255 correspondent à des caractères « graphiques » spéciaux. Le programme suivant vous permettra de faire apparaître les caractères correspondant à une suite de codes :

```
1 INPUT " RANGE "; N1, N2
2 FOR XX = N1 TO N2 : PRINT " ( "; CHR$(XX) ;
  " ) " ;: NEXT XX : PRINT : GOTO1
```

---

* Ou Espaces *N.d.t.*

Le programme demandera d'abord les deux nombres délimitant l'intervalle choisi et affichera ensuite les caractères correspondant aux codes compris dans l'intervalle en question. Chaque caractère sera placé entre parenthèses.

Jeux de Caractères ASCII

CODE	CHAR	CODE	CHAR	CODE	CHAR	CODE	CHAR
0	NUL	32[1]		64	@	96[5]	´
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39[2]	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	TAB	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44[3]	,	76	L	108	l
13	CR	45	—	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125[6]	}
30	RS	62	>	94	↑	126	~
31	US	63	?	95[4]	←	127[7]	RUBOUT

[1]espace
[2]apostrophe
[3]virgule
[4]ou soulignage
[5]accent
[6]ou ALT MODE
[7]ou DEL

Index

La bibliothèque SYBEX

INTRODUCTION AU PASCAL, Pierre Le Beux

500 pages, Réf. PA01
Un livre complet et progressif pour l'apprentissage du Pascal. Depuis les concepts de base jusqu'aux programmes élaborés et les traitements de fichiers et graphiques.

LE PASCAL PAR LA PRATIQUE, Pierre Le Beux et Henri Tavernier

550 pages, Réf. PA02
Plus de 140 exercices complètement traités : énoncé du problème, algorithmes et commentaires, programmes, exemples d'exécution.

LE GUIDE DU PASCAL, Jacques Tiberghien

450 pages, Réf. PA03
Le dictionnaire encyclopédique complet du PASCAL, définissant chaque mot, procédure, fonction des différentes versions du PASCAL.

INTRODUCTION AU BASIC, Pierre Le Beux

335 pages, Réf. PB02
Une introduction progressive au BASIC, couvrant tous les aspects du langage actuellement disponibles sur microordinateurs et systèmes de temps partagé.

LE BASIC PAR LA PRATIQUE : 60 EXERCICES, Jean-Pierre Lamoitier

200 pages, Réf. PB01, 2e édition
Les exercices, complètement traités (énoncé, analyse, solution avec ordinogramme, programme d'application) permettent au lecteur de perfectionner très rapidement ses connaissances en BASIC. Tous les programmes sont en BASIC Microsoft.

AU CŒUR DES JEUX EN BASIC, Richard Mateosian

300 pages, Réf. PB03
Enseigne la programmation du Basic par les jeux. Ces jeux sont en Basic Microsoft et peuvent être utilisés sur TRS80, APPLE II et PET/CBM.

LE BASIC POUR L'ENTREPRISE, Xuan Tung BUI

172 pages, Réf. PB04
Ce livre présente toutes les méthodes importantes de gestion en expliquant leur but, leur principe ainsi que leur réalisation en Basic. Il est constitué d'un choix de programmes testés, et prêts à être utilisés pour la décision d'entreprise.

VOTRE PREMIER ORDINATEUR, Rodnay Zaks

290 pages, Réf. C1B
Une introduction simple et complète aux petits ordinateurs avec leurs périphériques. Ce qu'ils peuvent faire et comment se les procurer.

PROGRAMMATION DU Z80, Rodnay Zaks
600 pages, Réf. C780
Une introduction complète à la programmation en langage assembleur pour le Z80.

GUIDE DU CP/M AVEC MP/M, Rodnay Zaks
420 pages, Réf. C8
Un guide de référence indispensable sur le CP/M, système d'exploitation standard le plus utilisé sur les microordinateurs.

LES MICROPROCESSEURS, Rodnay Zaks et Pierre Le Beux
320 pages, Réf. C4
Tous les concepts et techniques liés aux microprocesseurs, depuis les principes de base jusqu'à la programmation. Une introduction simple et complète.

TECHNIQUES D'INTERFACE, Austin Lesea et Rodnay Zaks
416 pages, Réf. C5, 3e édition
Exposé clair et systématique sur les méthodes et les composants qui permettent de construire un système complet.

PROGRAMMATION DU 6502, Rodnay Zaks
300 pages, Réf. C3, 2e édition
La programmation des systèmes basés sur le microprocesseur 6502, des concepts de base aux structures de données les plus avancées.

APPLICATION DU 6502, Rodnay Zaks
300 pages, Réf. D802
Techniques d'applications pratiques : le livre des entrées-sorties du 6502.

PROGRAMMATION DU 6800, Daniel Jean David et Rodnay Zaks
380 pages, Réf. C6
Pour apprendre à programmer le 6800, avec de nombreux exercices. Un véritable apprentissage par l'action.

LEXIQUE INTERNATIONAL MICROPROCESSEURS, avec dictionnaire abrégé en 10 langues
192 pages, Réf. C2-B
Lexique de poche contenant toutes les définitions et les sigles du langage informatique avec, en plus, un dictionnaire abrégé des principaux termes en 10 langues.

VERSION ANGLAISE DES TITRES EN FRANÇAIS :

Your First Computer, réf. C200A
Programming the Z80, réf. C280
CP/M Handbook, réf. C300
6502 Applications Book, réf. D302
Inside Basic Games, réf. B245
Fifty BASIC Exercices, réf. B250
Pascal Handbook, réf. P320
From chips to systems : an introduction to microprocessors, réf. C201A
Microprocessor Interfacing Techniques, réf. C207
Programming the 6502, réf. C202
International Microcomputer dictionary, réf. X2

EN ANGLAIS,

INTRODUCTION TO WORD PROCESSING, Hal Glatzer

300 pages, Réf. W101
Un livre simple et clair sur le traitement de texte : ses possibilités, son utilisation, ses avantages. Quelle solution envisager selon vos besoins.

INTRODUCTION TO WORDSTAR, Arthur Naiman

200 pages, Réf. W110
Apprenez facilement à utiliser le Wordstar, un programme de traitement de texte hautement performant pout tout utilisateur d'ordinateur.

INTRODUCTION TO PASCAL (Including UCSD PASCAL), Rodnay Zaks

440 pages, Réf. P310
Une introduction progressive au langage Pascal (Standard et UCSD). Aucune connaissance technique n'est requise.

PASCAL PROGRAMS FOR SCIENTISTS AND ENGINEERS, Alan Miller

250 pages, Réf. P340
Une présentation complète des algorithmes les plus fréquemment utilisés pour les applications scientifiques et techniques.

FIFTY PASCAL PROGRAMS, Rodnay Zaks et Rudolph Langer

275 pages, Réf. P350
50 programmes en PASCAL, des applications mathématiques aux applications commerciales, en passant par des programmes de jeux. Explique les techniques de programmation et permet d'acquérir une véritable pratique.

APPLE PASCAL GAMES, Douglas Hergert et Joseph T. Kalash

350 pages, Réf. P360
L'ensemble des jeux les plus populaires en PASCAL UCSD : criblage, horserece, Keno, overunder, baccarat, kismet, blackbox, chuckaluck... Pratiquez le PASCAL en vous amusant !

BASIC PROGRAMS FOR SCIENTISTS AND ENGINEERS, Alan Miller

345 pages, Réf. B240
Second volume de la série SYBEX « Programmes pour Scientifiques et Ingénieurs ». Un jeu complet des algorithmes scientifiques les plus importants, et leur implémentation en BASIC.

INTRODUCTION TO UCSD PASCAL SYSTEMS, Charles W. Grant et Jon Butah

250 pages, Réf. P370
Une introduction simple et claire à l'utilisation des systèmes opératoires en UCSD PASCAL.

DON'T (or How to Care for Your Computer), Rodnay Zaks

244 pages, Réf. C400
Comment se servir correctement de son ordinateur y compris ce qu'il faut faire quand il ne fonctionne plus. Le manuel détaillé des consignes à respecter, pour tous les publics.

PROGRAMMING THE Z8000, Richard Mateosian

312 pages, Réf. C281
Comment programmer le Z8000. Couvre en détail l'architecture et le fonctionnement du Z8000.

6502 GAMES, Rodnay Zaks

304 pages, Réf. G402
Troisième livre de la série 6502. Enseigne les techniques de programmation les plus avancées en utilisant les jeux comme support pédagogique.

A MICROPROGRAMMED APL IMPLEMENTATION, Rodnay Zaks

256 pages, Réf. Z10
Ce livre, pour spécialistes, présente une analyse complète de l'interpréteur APL avec listings de microprogrammes.

NOS LIVRES SONT DISPONIBLES EN FRANÇAIS, ANGLAIS, ALLEMAND, ESPAGNOL, ITALIEN, SUÉDOIS, JAPONAIS, BULGARE, RUSSE.

POUR UN CATALOGUE COMPLET DE NOS PUBLICATIONS

EUROPE
4, place Félix-Éboué
75583 Paris Cedex 12
Tél. : (1) 347.30.20
Télex : 211801



U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tél. : (415) 848.8233
Télex : 336311

ALLEMAGNE
Heyestr. 22
4000 Düsseldorf 12
Tél. : (0211) 287066
Télex : 08588163

Imprimé en France. — Imprimerie JOUVE, 18, rue Saint-Denis, 75001 PARIS
Dépôt légal : février 1982 — Imprimeur : 9551

Créez vos propres programmes de jeu avec

AU CŒUR DES JEUX EN BASIC

• Huit catégories de jeux d'ordinateur sont décrites, expliquées et analysées en détail, pour vous apprendre à concevoir des programmes élégants et sans erreurs.

• Chaque caractéristique du Basic, détaillée dans chaque chapitre, vous aide à réaliser vous-même des jeux d'ordinateur en Basic Interactif.

• Les programmes finaux sont rédigés en Basic Microsoft avec versions spécifiques pour PET/CBM, TRS-80 et APPLE II.

Apprenez à concevoir des jeux en Basic Interactif, en vous amusant !

L'AUTEUR

Richard Mateosian s'est intéressé à la conception des petits ordinateurs dès leur apparition dans les années 60.

Docteur en mathématiques de l'Université de Berkeley, il est conférencier à l'Université de San Francisco et conseil en informatique.

ISBN 2-902414-33-1


SYBEX
AU CŒUR
DES JEUX EN
BASIC
MATEOSIAN



Document numérisé avec amour par :

AMSTRAD CPC

MÉMOIRE ÉCRITE

ACME

https://acpc.me/